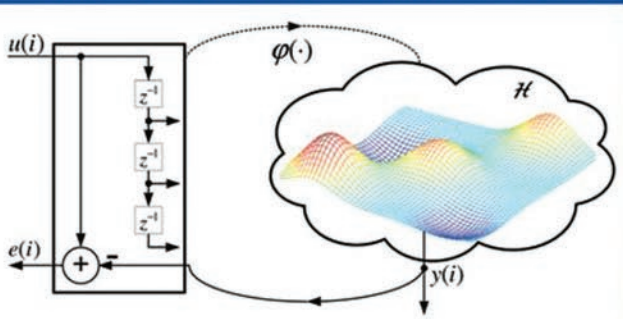
Wiley Series in Adaptive and Learning Systems for Signal Processing, Communications, and Control
Simon Haykin, Series Editor

Kernel Adaptive Filtering

A COMPREHENSIVE INTRODUCTION



WEIFENG LIU
JOSÉ C. PRÍNCIPE
SIMON HAYKIN

WILEY

KERNEL ADAPTIVE FILTERING

Adaptive and Learning Systems for Signal Processing, Communication, and Control

Editor: Simon Haykin

A complete list of titles in this series appears at the end of this volume.

KERNEL ADAPTIVE FILTERING

A Comprehensive Introduction

Weifeng Liu, José C. Príncipe, and
Simon Haykin

WILEY

JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data:

Liu, Weifeng
Kernel adaptive filtering : a comprehensive introduction / Jose C. Principle, Weifeng Liu, Simon Haykin.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-470-44753-6
1. Adaptive filters. 2. Kernel functions. I. Príncipe, José C. II. Haykin, Simon S., 1931– III. Title.
TK7872.F5P745 2010
621.382′23–dc22

2009042654

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

To our families

CONTENTS

PREFACE

For the first time, this book presents a comprehensive and unifying introduction to kernel adaptive filtering. Adaptive signal processing theory has been built on three pillars: the linear model, the mean square cost, and the adaptive least-square learning algorithm. When nonlinear models are required, the simplicity of linear adaptive filters evaporates and a designer has to deal with function approximation, neural networks, local minima, regularization, and so on. Is this the only way to go beyond the linear solution? Perhaps there is an alternative, which is the focus of this book. The basic concept is to perform adaptive filtering in a linear space that is related nonlinearly to the original input space. If this is possible, then all three pillars and our intuition about linear models can still be of use, and we end up implementing nonlinear filters in the input space.

This book will draw on the theory of reproducing kernel Hilbert spaces (RKHS) to implement the nonlinear transformation of the input to a high-dimensional feature space induced by a positive-definite function called *reproducing kernel*. If the filtering and adaptation operations to be performed in RKHS can be expressed by inner products of projected samples, then they can be directly calculated by kernel evaluations in the input space. We use this approach to introduce a family of adaptive filtering algorithms in RKHS:

- The kernel least-mean-square algorithm
- The kernel affine projection algorithms
- The kernel recursive least-squares algorithm
- The extended kernel recursive least-squares algorithm

These kernel-learning algorithms bridge closely two important areas of adaptive filtering and neural networks, and they embody beautifully two important methodologies of error-correction learning and memory-based learning. The bottlenecks of the RKHS approach to nonlinear filter design are the need for regularization, the need to select the kernel function, and the need to curtail the growth of the filter structure. This book will present in a mathematically rigorous manner the issues and the solutions to all these

problems, and it will illustrate with examples the performance gains of kernel adaptive filtering.

Chapter 1 starts with an introduction to general concepts in machine learning, linear adaptive filters, and conventional nonlinear methods. Then, the theory of reproducing kernel Hilbert spaces is presented as the mathematical foundation of kernel adaptive filters. We stress that kernel adaptive filters are universal function approximators, have no local minima during adaptation, and require reasonable computational resources.

Chapter 2 studies the kernel least-mean-square algorithm, which is the simplest among the family of kernel adaptive filters. We develop the algorithm in a step-by-step manner and delve into all the practical aspects of selecting the kernel function, picking the step-size parameter, sparsification, and regularization. Two computer experiments, one with Mackey–Glass chaotic time-series prediction and the other with nonlinear channel equalization, are presented.

Chapter 3 covers the kernel affine projection algorithms, which is a family of four similar algorithms. The mathematical equations of filtering and adaptation are thoroughly derived from first principles, and useful implementation techniques are discussed fully. Many well-known methods can be derived as special cases of the kernel affine projection algorithms. Three detailed applications are included to show their wide applicability and design flexibility.

Chapter 4 presents the kernel recursive least-squares algorithm and the theory of Gaussian process regression. A sparsification approach called approximate linear dependency is discussed. And with the aid of the Bayesian interpretation, we also present a powerful model selection method called "maximum marginal likelihood". Two computer experiments are conducted to study the performance of different sparsification schemes and the effectiveness of maximum marginal likelihood to determine the kernel parameters.

Chapter 5 discusses the extended kernel recursive least-squares algorithm on the basis of the kernel recursive least-squares algorithm. We study systematically the problem of estimating the state of a linear dynamic system in RKHS from a sequence of noisy observations. Several important theorems are presented with proofs to outline the significance and basic approaches. This chapter contains two examples, Rayleigh channel tracking and Lorenz time-series modeling.

Chapter 6 is devoted to addressing the principal bottleneck of kernel adaptive filters, i.e., their growing structure. We introduce a subjective information measure called *surprise* and present a unifying sparsification scheme to curtail the growth effectively of kernel adaptive filters. Three interesting computer simulations are presented to illustrate the theories.

This book should appeal to engineers, computer scientists, and graduate students who are interested in adaptive filtering, neural networks, and kernel methods. A total of 12 computer-oriented experiments are distributed throughout the book that have been designed to reinforce the concepts discussed in the chapters. The computer experiments are listed in Table 1. Their MATLAB®

Table 1. A listing of all computer experiments in the book. MATLAB® programs that generate the results can be downloaded by all readers from the book's website http://www.cnel.ufl.edu/~weifeng/publication.htm.

Computer experiment	Topic
2.1	KLMS Applied to Mackey–Glass Time-Series Prediction
2.2	KLMS Applied to Nonlinear Channel Equalization
3.1	KAPA Applied to Mackey–Glass Time-Series Prediction
3.2	KAPA Applied to Noise Cancellation
3.3	KAPA Applied to Nonlinear Channel Equalization
4.1	KRLS Applied to Mackey–Glass Time-Series Prediction
4.2	Model Selection by Maximum Marginal Likelihood
5.1	EX-KRLS Applied to Rayleigh Channel Tracking
5.2	EX-KRLS Applied to Lorenz Time-Series Prediction
6.1	Surprise Criterion Applied to Nonlinear Regression
6.2	Surprise Criterion Applied to Mackey–Glass Time-Series Prediction
6.3	Surprise Criterion Applied to CO_2 Concentration Forecasting

implementations can be downloaded directly from the website http://www.cnel.ufl.edu/~weifeng/publication.htm. To keep the codes readable, we placed simplicity over performance during design and implementation. These programs are provided without any additional guarantees.

We have strived to reflect fully the latest advances of this emerging area in the book. Each chapter concludes with a summary of the state of the art and potential future directions for research. This book should be a useful guide to both those who look for nonlinear adaptive filtering methodologies to solve practical problems and those who seek inspiring research ideas in related areas.

ACKNOWLEDGMENTS

We would like to start by thanking Dr. Puskal P. Pokharel, Seagate Technology; Dr. Murali Rao, University of Florida; Dr. Jay Gopalakrishnan, University of Florida; and Il Park, University of Florida, for their help in the development of the kernel adaptive filtering theory. We are most grateful to Dr. Aysegul Gunduz, Albany Medical College, Wadsworth Center; Dr. John Harris, University of Florida; and Dr. Steven Van Vaerenbergh, University of Cantabria, Spain, for providing many useful comments and constructive feedback on an early version of the manuscript of the book.

Many others have been kind enough to read critically selected chapters/sections of the book; in alphabetical order, they are as follows:

Erion Hasanbelliu, University of Florida, Gainesville, Florida

Dr. Kyu-hwa Jeong, Intel Corporation, Santa Clara, California

Dr. Ruijiang Li, University of California, San Diego, California

Dr. Antonio Paiva, University of Utah, Salt Lake City, Utah

Alexander Singh, University of Florida, Gainesville, Florida

Dr. Yiwen Wang, Hong Kong University of Science and Technology, Hong Kong

Dr. Jianwu Xu, University of Chicago, Chicago, Illinois

We also wish to thank (in alphabetical order): Dr. Peter Bartlett, University of California, Berkeley; Dr. Andrzej Cichocki, Riken, Brain Science Institute, Japan; Dr. Corinna Cortes, Google Lab; Dr. Graham C. Goodwin, University of Newcastle, UK; Dr. Michael Jordan, University of California, Berkeley; Dr. Thomas Kailath, Stanford University; Dr. Joel S. Kvitky, Rand Corporation; Dr. Yann LeCun, New York University; Dr. Derong Liu, University of Illinois at Chicago; Dr. David J. C. MacKay, University of Cambridge, UK; Dr. Tomaso Poggio, Massachusetts Institute of Technology; Dr. Ali H. Sayed, University of California, Los Angeles; Dr. Bernhard Schölkopf, Max Planck Institute for Biological Cybernetics, Germany; Dr. Sergios Theodoridis, University of Athens, Greece; Dr. Yoram Singer, Google Lab; Dr. Alexander J. Smola, Yahoo! research; Dr. Johan Suykens, Katholieke Universiteit

Leuven, Belgium; Dr. Paul Werbos, The National Science Foundation; Dr. Bernard Widrow, Stanford University; and Dr. Chris Williams, University of Edinburgh, UK.

We thank the staff at Wiley, publisher George Telecki, editorial assistant Lucy Hitz, and production editor Kris Parrish, as well as the project manager, Stephanie Sakson from Toppan Best-Set Premedia, for their full support and encouragement in preparing the manuscript of the book and for all their behind-the-scenes effort in the selection of the book cover and the production of the book.

Last, but by no means least, we are grateful to Lola Brooks, McMaster University, for typing many sections of the manuscript.

NOTATION

The book discusses many algorithms involving various mathematical equations. A convenient and uniform notation is a necessity to convey clearly the basic ideas of the kernel adaptive filtering theory. We think it is helpful to summarize and explain at the beginning of the text our notational guidelines for ease of reference.

There are mainly *three* types of variables we need to distinguish:

scalar, vector, and matrix variables

The following is a list of the notational conventions used in the book:

1. We use *small italic* letters to denote *scalar variables*. For example, the output of a filter is a scalar variable, which is denoted by y.
2. We use *CAPITAL ITALIC* letters to denote *SCALAR CONSTANTS*. For example, the order of a filter is a scalar constant, which is denoted by L.
3. We use **small bold** letters for **vectors**.
4. We use **CAPITAL BOLD** letters to denote **MATRICES**.
5. We use *parentheses* to denote the *time dependency* of any variables (either scalar, vector, or matrix). For example, $d(i)$ means the value of a scalar d at time (or iteration) i. $\mathbf{u}(i)$ means the value of a vector $\mathbf{u}$ at time (or iteration) i. Similarly $\mathbf{G}(i)$ means the value of a matrix $\mathbf{G}$ at time (or iteration) i. There is no rule without an exception. f_i is used to denote the estimate of an input–output mapping f at time (or iteration) i since parenthesis is preserved for input argument like $f_i(\mathbf{u})$.
6. We use the superscript T to denote *transposition*. For example, if

$$\mathbf{d} = \begin{bmatrix} d(1) \\ d(2) \\ \cdots \\ d(N) \end{bmatrix}$$

then

$$\mathbf{d}^T = [d(1), d(2), \ldots, d(N)].$$

7. All variables in our presentation are *real*. We do not discuss complex numbers in this book.
8. All vectors in our presentation are *column vectors* without exception.
9. We use subscript indices to denote 1) a component of a vector (or a matrix), 2) a general vector that the index is not related to time (or iteration). For example, $\mathbf{c}_i$ could mean the ith vector in some set or the ith component of the vector $\mathbf{c}$ according to the context.

We have made every effort to make the notation consistent and coherent for the benefit of the reader. The following Table 2 summarizes and lists some typical examples.

Table 2. Notation.

	Description	Examples
Scalars	Small *italic* letters	d
Vectors	Small **bold** letters	$\mathbf{w}$, $\boldsymbol{\omega}$, $\mathbf{c}_i$
Matrices	Capital **BOLD** letters	$\mathbf{U}$, $\boldsymbol{\Phi}$
Time or iteration	Indices in parentheses	$\mathbf{u}(i)$, $d(i)$
Component of vectors/matrices	Subscript indices	$\mathbf{a}_j$, $\mathbf{G}_{i,j}$
Linear spaces	Capital mathbb letters	$\mathbb{F}$, $\mathbb{H}$
Scalar constants	Capital *ITALIC* letters	L, N

ABBREVIATIONS AND SYMBOLS

We collect here a list of the main abbreviations and symbols used throughout the text for ease of reference.

$(\cdot)^T$	vector or matrix transposition
$\mathbf{A}^{-1}$	inverse of matrix $\mathbf{A}$
$\mathbf{E}[\cdot]$	expected value of a random variable
$m(\cdot)$	the mean of a random variance
$\sigma^2(\cdot)$	the variance of a random variable
$<\cdot,\cdot>$	inner product
$\|\cdot\|$	norm of a vector; square root of the inner product with itself
$\|\cdot\|$	absolute value of a real number or determinant of a matrix
$\propto$	proportional to
$\sim$	distributed according to
∇	gradient
$\mathbf{0}$	zero vector or matrix
β	forgetting factor
$\mathcal{C}(i)$	dictionary or center set at iteration i
$d(i)$	desired output at time or iteration i (a real scalar)
diag$\{a, b\}$	a diagonal matrix with diagonal entries a and b
δ_1	distance threshold in novelty criterion
δ_2	prediction error threshold in novelty criterion
δ_3	threshold in approximate linear dependency test
δ_{ij}	Kronecker delta
$\Delta\mathbf{w}(i)$	weight adjustment at time or iteration i (a column vector in an Euclidean space)
$\mathcal{D}$	data set
$e(i)$	output estimation error at time or iteration i
$\mathbb{F}$	feature space induced by the kernel mapping
$\mathbf{G}$	Gram matrix of (transformed) input data
$\mathbb{H}$	reproducing kernel Hilbert space

$\mathbf{I}$	identity matrix
$J(i)$	error cost at time or iteration i
$\mathbf{k}(i)$	Kalman gain (or gain vector) at time or iteration i
$K(\mathbf{A})$	condition number of a matrix $\mathbf{A}$
$\kappa(\mathbf{u}, \mathbf{u}')$	kernel (or covariance) function evaluated at $\mathbf{u}$ and $\mathbf{u}'$
L	dimensionality of the input space
λ	regularization parameter
M	dimensionality of the feature space
$\mathcal{M}$	misadjustment of the least-mean-square algorithm
$\mathbf{n}(i)$	additive noise in the state space at time or iteration i
N	number of training data
η	step-size parameter
$O(\cdot)$	of the order of a number
$\mathbf{P}$	state-error correlation matrix
$\boldsymbol{\varphi}(\cdot)$	a mapping induced by a reproducing kernel
$\boldsymbol{\varphi}(i)$	transformed filter input at time or iteration i (a column vector in a feature space)
$\boldsymbol{\Phi}$	transformed input data matrix
$\mathbf{R}$	covariance matrix of (transformed) input data
$\mathbb{R}$	the set of real numbers
$\mathbb{R}^L$	L-dimensional real Euclidean space
$\varsigma_{\max}$	the maximum eigenvalue
$\text{tr}(\mathbf{A})$	trace of matrix $\mathbf{A}$
T_1	abnormality threshold in surprise criterion
T_2	redundancy threshold in surprise criterion
$\mathbf{u}(i)$	filter input at time or iteration i (a column vector in an Euclidean space)
$\mathbb{U}$	input domain
$\mathbf{U}$	input data matrix
$v(i)$	additive noise in the output at time or iteration i
$\mathbf{w}(i)$	weight estimate at time or iteration i (a column vector in an Euclidean space)
$\boldsymbol{\omega}(i)$	weight estimate at time or iteration i (a column vector in a feature space)
z^{-1}	unit delay operator

AIC	Akaike information criterion
ALD	approximate linear dependency
APA	affine projection algorithm
BIC	Bayesian information criterion
CC	coherence criterion
CV	cross-validation
ENC	enhanced novelty criterion
EX-RLS	extended recursive least squares algorithm
EX-KRLS	extended kernel recursive least squares algorithm

GPR	Gaussian process regression
LMS	least-mean-square algorithm
LOOCV	leave-one-out cross-validation
LS	least squares
MAP	maximum a posterior
MDL	minimum description length
MSE	mean square error
MML	maximum marginal likelihood
NC	novelty criterion
NLMS	normalized least-mean-square algorithm
KA	kernel ADALINE
KAPA	kernel affine projection algorithm
KLMS	kernel least-mean-square algorithm
KRLS	kernel recursive least-squares algorithm
PCA	principal components analysis
PDF	probability density function
RAN	resource allocating network
RBF	radial-basis function
RKHS	reproducing kernel Hilbert space
RLS	recursive least-squares algorithm
RN	regularization network
RNN	recurrent neural network
SC	surprise criterion
SNR	signal-to-noise ratio
SVD	singular value decomposition
SVM	support vector machine
SW-KRLS	sliding window kernel recursive least-squares algorithm

1

BACKGROUND AND PREVIEW

1.1 SUPERVISED, SEQUENTIAL, AND ACTIVE LEARNING

Learning is a process by which the free parameters and the topology of a neural network are adapted through a process of stimulation by the environment in which the network is embedded [Haykin, 2009].

Adjustments on the free parameters are well studied in the adaptive filtering and neural network fields, whereas adaptation of the topology still has much room for investigation. Traditionally, the growing and pruning techniques for multilayer perceptrons are viewed as a heuristic network designing tool rather than as an integral part of learning itself. However, learning a network topology may be equally important, if not more important, than adjusting the free parameters in the network, as exemplified in biological learning where new synaptic connections in the human brain can grow or die. If the environment is changing over time, then the design of an "optimal" network topology beforehand by conventional methods such as Akaike information criterion, Bayesian information criterion, and minimum description length is not possible.[1] Therefore, it will make more sense to adapt the network structure over time as part of the learning process.[2]

Learning may be performed in three basic ways:

- *Supervised learning*, which requires the availability of a collection of desired (target) responses.

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

- *Unsupervised learning*, which is performed in a self-organized manner, bypassing the need for a desired response.
- *Reinforcement learning*, which learns through a sequence of state–action–reward and has no explicit input–output available.

In this book, we focus on supervised learning. In conceptual terms, we may think there is a teacher (supervisor) who has knowledge of the environment, where that knowledge is represented by input–desired examples (training data). Well-known supervised learning rules include *error-correction learning* like the Widrow–Hoff rule [Widrow and Hoff, 1960] and *memory-based learning* exemplified by k nearest-neighbor classifiers [Cover and Hart, 1967] and radial-basis function networks [Broomhead and Lowe, 1988].

Historically, machine learning has focused on nonsequential learning tasks, where the training data set can be constructed a priori and learning stops once this set is duly processed. Despite its wide applicability, there are many situations where learning takes place over time. A learning task is *sequential* if the training examples become available over time, usually one at a time. A learning algorithm is *sequential* if, for any given training examples

$$\{\mathbf{u}(1), d(1)\}, \{\mathbf{u}(2), d(2)\}, \ldots, \{\mathbf{u}(i), d(i)\}, \ldots$$

it produces a sequence of *hypotheses*

$$\hbar(1), \hbar(2), \ldots \hbar(i), \ldots$$

such that $\hbar(i)$ depends not only on the previous hypothesis $\hbar(i-1)$, but also on the current example $\{\mathbf{u}(i), d(i)\}$ [Giraud-Carrier, 2000]. Note that sometimes one relaxes slightly the above definition to allow the next hypothesis to depend on the previous one and a small subset of new training examples (rather than a single one) to trade off complexity and performance. It has been long argued that learning involves the ability to improve performance over time [Simon, 1983]. Clearly, humans acquire knowledge over time and knowledge is constantly revised, based on newly acquired information. In the study of robotic and intelligent agents, one finds that sequential learning is a must to deal with complex operating environments, which are usually changing and unpredictable [Brazdil, 1991, Tan, 1993]. Moreover, sequential learning algorithms typically require less computation and memory resources to update the hypothesis, and these algorithms may be the only possibility for large applications.

Active learning is another important concept in machine learning. There are possibly two scenarios where active learning finds applications. In the first case, unlabeled data (input without target) are abundant, but labeling data are expensive. For example, it is easy to get raw speech samples for a speech recognizer, whereas labeling these samples is tedious. The idea of active learning here is to construct an accurate recognizer with significantly fewer labels than you would need in a conventional supervised learning framework. For

these problems, no desired signal (or label) is available to quantify the importance of the candidate data sample.

In the second scenario, a large amount of data have already been gathered in pairs of input and target, but a subset of data must be selected for efficient training and sparse representation. This kind of problem commonly occurs in kernel methods and Gaussian process (GP) modeling.[3] For example, in support vector machines, only data that are close to the boundary are important. The rationale is that training data are not equally informative. Especially in a sequential learning setting, depending on the hypothesis (state) of a learning system, the same data may convey a different amount of information. This is understandable by our daily experience. A message conveys the most amount of information when it is first perceived, and its amount of information diminishes with repeated presentations. These kinds of problems are termed "active data selection" or "sparsification" to distinguish them from the first scenario. We use active learning as a general term referring to both cases.

Active sequential learning is natural. For instance, human beings assess incoming data every day based on knowledge and then decide how much resource should be allocated to process it. Intuitively, for a learning machine, one can expect that after processing sufficient samples from the same source, there is little need to learn, because of redundancy. It has been demonstrated that active learning can significantly reduce the computational complexity with equivalent performance. And it can even provide a more accurate and more stable solution in some situations.[4]

1.2 LINEAR ADAPTIVE FILTERS

Linear adaptive filters built around a linear combiner are designed for sequential learning. When we speak of adaptive filters, we have in mind a class of filtering structures, which are equipped with a built-in mechanism that enables such a filter to adjust its free parameters automatically in response to statistical variations in the environment in which the filter operates. This capability is behind a wide range of applications of adaptive filters in such diverse fields as adaptive equalization in communication receivers, adaptive noise cancellation in active noise control, adaptive beamforming in radar and sonar, system identification, and adaptive control, just to mention a few.[5]

The traditional class of supervised adaptive filters rely on *error-correction learning* for their adaptive capability. To illustrate what we mean by this form of learning, consider the filtering structure depicted in Figure 1.1; a tapped-delay-line (transversal) is most commonly used as the filter, on which the adaptation is performed. The filter embodies a set of adjustable parameters (weights), which is denoted by the vector $\mathbf{w}(i-1)$, where i denotes discrete time. An input signal vector, $\mathbf{u}(i)$, applied to the filter at time i, produces the actual response $y(i)$. This actual response is compared with an externally supplied desired response $d(i)$ to produce the error signal $e(i)$. This error signal

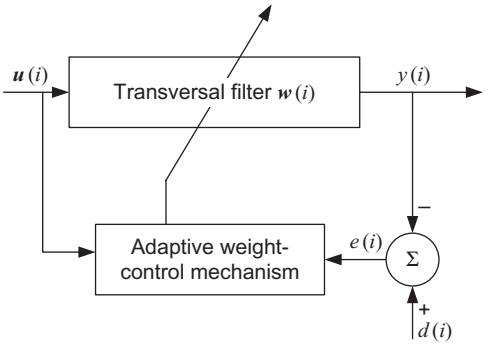


Figure 1.1. Basic structure of a linear adaptive filter.

is, in turn, used to produce an adjustment to the parameter vector $\mathbf{w}(i-1)$ of the filter by an incremental amount denoted by $\Delta\mathbf{w}(i)$. Accordingly, the *updated* parameter vector of the filter assumes the new value

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \Delta\mathbf{w}(i) \tag{1.1}$$

On the next iteration at time $i+1$, $\mathbf{w}(i)$ becomes the latest value of the parameter vector to be updated. The adaptive filtering process is continually repeated in this manner until the filter reaches a condition, whereafter the parameter adjustments become small enough to stop the adaptation. As is clear, the weights here embody the hypothesis in the definition of sequential learning.

Starting from some initial condition denoted by $\mathbf{w}(0)$, the *ensemble-averaged square error*

$$J(i) = \mathbf{E}\left[e^2(i)\right], \quad i = 1, 2, \cdots \tag{1.2}$$

plotted versus time i, traces the *learning curve* of the adaptive filtering process. The expectation in equation (1.2), which is denoted by $\mathbf{E}[\cdot]$, is carried out for an ensemble of different training sets.

An important issue in the design of adaptive filters is to ensure that the learning curve is *convergent* with an increasing number of iterations. Under this condition, we may define the speed of adaptation as the number of iterations i needed for the ensemble-averaged square error, $J(i)$, to reach a relatively "steady-state" value. We may then say that the adaptive filter is *convergent in the mean-square-error sense*.

1.2.1 Least-Mean-Square Algorithm

The simplest and most commonly used form of an adaptive filtering algorithm is the so-called *least-mean-square* (LMS) *algorithm*. Basically, the LMS algorithm operates by minimizing the instantaneous cost function

$$J(i) = \frac{1}{2} e^2(i) \tag{1.3}$$

where the factor 1/2 is introduced to simplify the mathematical formulation. Given that the parameter vector of the filter is $\mathbf{w}(i-1)$, the *prediction error* $e(i)$ is defined by

$$e(i) = d(i) - \mathbf{w}(i-1)^T \mathbf{u}(i) \tag{1.4}$$

Correspondingly, the *instantaneous gradient vector* is given by

$$\frac{\partial}{\partial \mathbf{w}(i-1)} J(i) = -e(i)\mathbf{u}(i) \tag{1.5}$$

Following the instantaneous version of the *method of gradient descent*, the adjustment $\Delta\mathbf{w}(i)$ applied to the algorithm at time i is defined by

$$\Delta\mathbf{w}(i) = \eta e(i)\mathbf{u}(i) \tag{1.6}$$

where η is the *step-size parameter*. Thus, using equation (1.6) in equation (1.1) yields the following update rule for the filter's parameter vector:

$$\underbrace{\mathbf{w}(i)}_{\text{Updated estimate}} = \underbrace{\mathbf{w}(i-1)}_{\text{Old estimate}} + \underbrace{\eta e(i)\mathbf{u}(i)}_{\text{Adjustment}} \tag{1.7}$$

where the prediction error $e(i)$ is itself defined in equation (1.4).

Examining the computations described above, we readily view the basic *simplicity* of the LMS algorithm. Nonetheless, the algorithm can deliver an effective performance, provided that the step-size parameter η is properly chosen. Most importantly, the LMS algorithm is *model independent*, in that no structural restriction was imposed on how the training data were generated. Consequently, the LMS algorithm is known for its robustness. For best performance, the step-size parameter η should be assigned a relatively small value. However, from a practical perspective, such a choice has a serious disadvantage: A small η makes the LMS algorithm converge slowly.

1.2.2 Recursive Least-Squares Algorithm

To overcome the slow rate of adaptation of the LMS algorithm, we may look to another adaptive filtering algorithm: the recursive least-squares (RLS) algorithm.

In a sense, the RLS algorithm follows a rationale similar to the LMS algorithm, in that they are both examples of error-correction learning but with a basic difference:

At every iteration i, the LMS algorithm aims at minimizing the instantaneous value of the squared estimation error $J(i)$ as in equation (1.3), the RLS algorithm aims at minimizing the sum of squared estimation errors up to and including the current time i.

Mathematically, the cost function for the RLS algorithm is defined by

$$J(i) = \sum_{j=1}^{i} \left[d(j) - \mathbf{w}^T \mathbf{u}(j) \right]^2 \tag{1.8}$$

Typically, the RLS algorithm converges to a relatively stable condition an order of magnitude faster than the LMS algorithm. However, this improvement in performance is achieved at some cost, as discussed later in this section.

Considering the operation of the RLS algorithm at time i, let $\mathbf{w}(i-1)$ denote the old estimate of the parameter vector computed at the preceding time instant $i-1$. In an information-theoretic sense, the *state* of the adaptive filter, which is symbolized by the estimate $\mathbf{w}(i-1)$, provides a summary of all the data processed by the RLS algorithm, starting from the initial condition $i = 0$ up to and including time $i-1$. We may, therefore, view the prediction error

$$e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i) \tag{1.9}$$

as the new information supplied to the algorithm at time i by the pair of input vector and desired response: $\{\mathbf{u}(i), d(i)\}$. Indeed, it is in light of this statement that the prediction error is referred to as *innovation*.

In the course of working through the derivation of the RLS algorithm, we find that the adjustment supplied to the old estimate $\mathbf{w}(i-1)$ is now defined by

$$\Delta\mathbf{w}(i) = \mathbf{k}(i)e(i) \tag{1.10}$$

where $\mathbf{k}(i)$ is called the *gain vector* of the RLS algorithm. To be specific, $\mathbf{k}(i)$ is defined by

$$\mathbf{k}(i) = \mathbf{P}(i)\mathbf{u}(i) \tag{1.11}$$

where $\mathbf{P}(i)$ is the *state-error correlation matrix*. In fact, the matrix $\mathbf{P}(i)$ is the inverse of the time-averaged correlation matrix, $\mathbf{R}(i)$, of the input vector $\mathbf{u}(i)$, as shown by

$$\mathbf{P}(i) = \mathbf{R}^{-1}(i) \tag{1.12}$$

with $\mathbf{R}(i)$ itself being defined by

$$\mathbf{R}(i) = \sum_{j=1}^{i} \mathbf{u}(j)\mathbf{u}^T(j) \tag{1.13}$$

Accordingly, the updated estimate of the actual parameter vector $\mathbf{w}(i)$ in the RLS algorithm is defined by

$$\underbrace{\mathbf{w}(i)}_{\text{Updated estimate}} = \underbrace{\mathbf{w}(i-1)}_{\text{Old estimate}} + \underbrace{\mathbf{k}(i)e(i)}_{\text{Adjustment}} \tag{1.14}$$

In closing this brief discussion of the LMS and RLS algorithms, we may compare their individual characteristics as follows:

1. Computational complexity of the LMS algorithm scales linearly with the dimension of the parameter vector $\mathbf{w}$, whereas the computational complexity of the RLS algorithm follows a square law.
2. Being model independent, the LMS algorithm is typically more robust than the RLS algorithm.
3. The convergence rate of the RLS algorithm is typically an order of magnitude faster than the LMS algorithm.
4. Last, but by no means least, the LMS algorithm propagates the estimation error from one iteration to the next, whereas the RLS algorithm propagates the error-covariance matrix. This statement is a further testimony to the simplicity of the LMS algorithm.

1.2.3 Extended Recursive Least-Squares Algorithm

The extended recursive least-squares (EX-RLS) algorithm is a special case of a more general algorithm called the *Kalman filter* [Kalman, 1960, Haykin, 2002]. Compared with RLS and LMS, a distinctive feature of the extended recursive least-squares algorithm is that its mathematical formulation is described in terms of *state-space concepts*. Although RLS also embodies the concept of state, the state in RLS is time invariant. Consider a linear dynamic system described by the signal-flow graph shown in Figure 1.2. The state vector

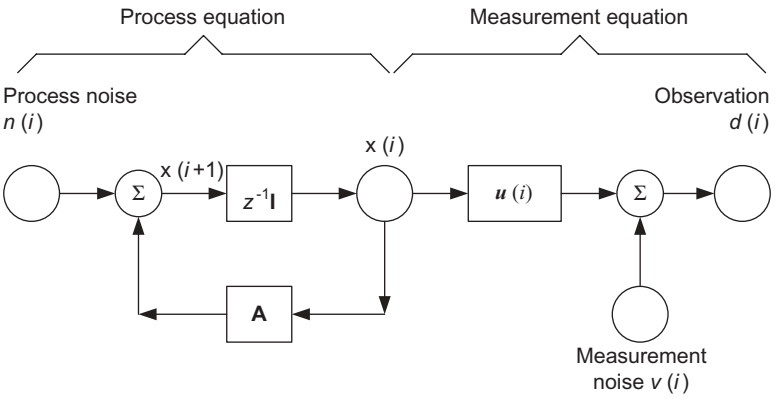


Figure 1.2. Signal-flow graph representation of a linear dynamic system.

denoted by $\mathbf{x}(i)$ in Figure 1.2 is defined as the minimal set of data that is sufficient to describe the unforced dynamic behavior of the system uniquely. In other words, the state comprises the fewest data on the past of the system that are needed to predict its future behavior. Typically, the state $\mathbf{x}(i)$ is unknown and we need to use a set of observations to estimate it.

In mathematical terms, the signal-flow graph of Figure 1.2 embodies the following pair of equations:

1. *A process equation*

$$\mathbf{x}(i+1) = \mathbf{A}\mathbf{x}(i) + \mathbf{n}(i) \tag{1.15}$$

In this equation, the L-by-1 vector $\mathbf{n}(i)$ represents *process noise*, modeled as a zero-mean, white-noise process whose correlation matrix is defined by

$$\mathbf{E}\left[\mathbf{n}(i)\mathbf{n}(j)^T\right] = \begin{cases} \mathbf{Q}_1, & i = j \\ \mathbf{0}, & i \neq j \end{cases} \tag{1.16}$$

The process equation (1.15) models an unknown physical stochastic phenomenon denoted by the L-by-1 state vector $\mathbf{x}(i)$ as the output of a linear dynamic system excited by the white noise $\mathbf{n}(i)$, as depicted in the left-hand portion of Figure 1.2. The linear dynamic system is uniquely characterized by the feedback connection of two units: the L-by-L *transition matrix*, denoted by $\mathbf{A}$, and the *memory unit*, denoted by $z^{-1}\mathbf{I}$, where z^{-1} is the unit time delay and $\mathbf{I}$ is the L-by-L identity matrix. In general, $\mathbf{A}$ can be time variant.

2. *A measurement equation*, which describes the observation as

$$d(i) = \mathbf{u}(i)^T \mathbf{x}(i) + v(i) \tag{1.17}$$

where $\mathbf{u}(i)$ is known and called a *measurement vector* in this setting. The *measurement noise* is modeled as a zero-mean, white noise process with a fixed variance q_2. The measurement equation (1.17) relates the observable output of the system $d(i)$ to the state $\mathbf{x}(i)$, as depicted in the right-hand portion of Figure 1.2. Equation (1.17) bears some resemblance to the linear regression setting in RLS with $\mathbf{u}(i)$ as the input and $d(i)$ as the output. The distinctive difference is that the input–output mapping $\mathbf{x}(i)$ in equation (1.17) is *time variant* and governed by the process equation (1.15), whereas in RLS, it is *time invariant*.

It is assumed that $\mathbf{x}(1)$, which is the initial value of the state, is uncorrelated with both $\mathbf{n}(i)$ and $v(i)$ for $i \geq 1$. Two noise processes $\mathbf{n}(i)$ and $v(i)$ are statistically independent. Parameters $\mathbf{A}$, $\mathbf{Q}_1$, and q_2 are known a priori. With all those assumptions, the EX-RLS algorithm is required to solve the following problem:

Use the entire observations $\{\mathbf{u}(1), d(1)\}$, $\{\mathbf{u}(2), d(2)\}$, …, $\{\mathbf{u}(i), d(i)\}$, to find the minimum mean-square estimate of the state $\mathbf{x}(i+1)$.

We use $\mathbf{w}(i-1)$ to denote the optimal estimate of $\mathbf{x}(i)$ given the observations starting at time $j = 1$ and extending up to and including time $i-1$, i.e., $\{\mathbf{u}(1), d(1)\}$, $\{\mathbf{u}(2), d(2)\}$, …, $\{\mathbf{u}(i-1), d(i-1)\}$, and $\mathbf{w}(i)$ to denote the optimal estimate of $\mathbf{x}(i+1)$ given the observations starting at time $j = 1$ and extending up to and including time i. As in RLS, we define the *innovations process* associated with $d(i)$ as

$$e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i) \tag{1.18}$$

where $e(i)$ is the prediction error for the observed $d(i)$ and represents the new information in $d(i)$. The variance of the innovations process $e(i)$ is defined by

$$r(i) = \mathbf{E}\left[e(i)^2\right] = \mathbf{u}(i)^T \mathbf{P}(i-1)\mathbf{u}(i) + q_2 \tag{1.19}$$

where $\mathbf{P}(i-1)$ is the *state-error correlation matrix*. To proceed further, we need to introduce an important concept called *Kalman gain* or simply gain vector here, and a fundamental relation between the current optimal estimate and the previous optimal estimate

$$\underbrace{\mathbf{w}(i)}_{\text{Updated estimate}} = \mathbf{A}\,\underbrace{\mathbf{w}(i-1)}_{\text{Old estimate}} + \underbrace{\mathbf{k}(i)e(i)}_{\text{Adjustment}} \tag{1.20}$$

This equation shows that we can compute the minimum mean-square estimate $\mathbf{w}(i)$ of the state of a linear dynamic system by adding to the previous estimate $\mathbf{w}(i-1)$, which is premultiplied by the transition matrix $\mathbf{A}$, a correction term equal to $\mathbf{k}(i)e(i)$. The correction term equals the prediction error $e(i)$ premultiplied by the gain vector $\mathbf{k}(i)$. This equation is similar to equation (1.14) except for the step of premultiplying the estimate $\mathbf{w}(i-1)$ by the transition matrix $\mathbf{A}$. Furthermore, we can express the gain vector $\mathbf{k}(i)$ as

$$\mathbf{k}(i) = \mathbf{A}\mathbf{P}(i-1)\mathbf{u}(i)/r(i) \tag{1.21}$$

The problem remains of finding a recursive way of computing the state-error correlation matrix $\mathbf{P}(i-1)$. By using the famous *Riccati equation*, we have

$$\mathbf{P}(i) = \mathbf{A}\left[\mathbf{P}(i-1) - \mathbf{A}^{-1}\mathbf{k}(i)\mathbf{u}(i)^T\mathbf{P}(i-1)\right]\mathbf{A}^T + \mathbf{Q}_1 \tag{1.22}$$

Therefore, by initializing $\mathbf{w}(0) = \mathbf{0}$ and $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$, we can compute $r(1)$, $\mathbf{k}(1)$ and then update $\mathbf{w}(1)$, $\mathbf{P}(1)$ with $\{\mathbf{u}(1), d(1)\}$ supplied. The recursion can go on as long as observations are available. Here, λ is a positive number called the *regularization parameter*, which will be discussed in the subsequent chapters.

As mentioned, EX-RLS is a special case of a Kalman filter that is used in a wide range of engineering applications from radar to aerospace engineering, and it is one of the foundations in modern control theory and control systems engineering. In addition, EX-RLS provides a unifying framework for the derivation of the family of RLS filters using the state-space model. For example, by setting $\mathbf{A} = \beta^{-1/2}\mathbf{I}$ and ignoring $\mathbf{n}(i)$, we have the exponentially weighted RLS algorithm. Furthermore, by setting $\beta = 1$, we have the RLS algorithm itself.

Despite the simple structure of the linear adaptive filters (and probably because of it), they enjoy wide applicability and successes in diverse fields such as communications, control, radar, sonar, seismology, and biomedical engineering, among others. The theory of linear adaptive filters has reached a highly mature stage of development [Haykin, 2002]. However, the same cannot be said about nonlinear adaptive filters, as discussed next.

1.3 NONLINEAR ADAPTIVE FILTERS

The limited computational power of linear learning machines was highlighted by Minsky and Papert [1969] in their famous work on *Perceptrons*. In general, complex real-world applications require more expressive hypothesis spaces than linear functions. Now suppose we lift the linearity assumption, and the goal is to learn a continuous arbitrary input–output mapping $f : \mathbb{U} \rightarrow \mathbb{R}$ based on a sequence of examples. Apparently the problem of designing a nonlinear adaptive filter is much harder.[6] One simple way to implement a nonlinear adaptive filter is by cascading a static nonlinearity with a linear filter such as in Hammerstein and Wiener models [Wiener, 1958, Billings and Fakhouri, 1982]. However, in this approach, the modeling capability is limited, the choice of the nonlinearity is highly problem dependent, and there are local minima during training. Gabor [1968] tried using the *Volterra series*[7] to bypass the mathematical difficulties of nonlinear adaptive filtering, whereas it is clear that the complexity of the Volterra series explodes exponentially as its modeling capacity increases. An improvement of the Volterra series is the Wiener series, but slow convergence and high complexity still hinder its wide application. Later on, time-lagged multilayer perceptrons, radial-basis function networks, or recurrent neural networks were used to replace the linear combiner and trained in a real-time fashion by stochastic gradient [Lang and Hinton, 1988, Príncipe et al., 1992] (see Figure 1.3). They have a history of successes, but their nonconvex optimization nature prevents their widespread use in online applications.

Other forms of sequential learning can be found in the *Bayesian learning* literature [Winkler, 2003]. Recursive Bayesian estimation is a general probabilistic approach for estimating an unknown probability density function recursively over time using incoming measurements and a mathematical process model. It has a close connection to the Kalman filter in the adaptive

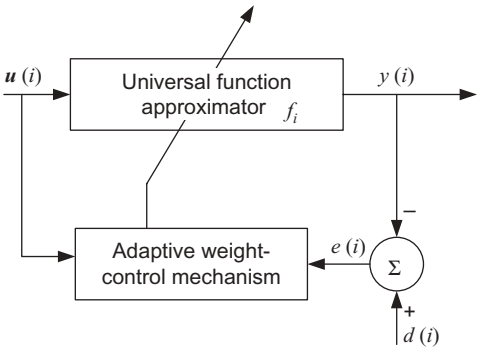


Figure 1.3. Basic structure of a nonlinear adaptive filter.

filtering theory [Kalman, 1960, Roweis and Ghahramani, 1999] but is usually complicated for arbitrary data distributions exemplified by sequential Monte Carlo methods [Doucet et al., 2000a].

Reinforcement learning [Sutton and Barto, 1998] is another area where sequential learning prevails. Reinforcement learning differs from supervised learning in several ways. The most important difference is that there is no presentation of input–output examples. Instead, after choosing an action based on the current state, the algorithm has access to an immediate reward and the subsequent state, but it is not told which action would have been in its best long-term interest. Online performance is crucial in reinforcement learning because the system must act on the environment to evaluate actions (i.e., action evaluation is concurrent with learning). Simply stated, the system learns through a process of reward and punishment without needing to receive a specification of how the task is to be achieved.

In this book, we follow a different course by focusing on a family of nonlinear adaptive filtering algorithms, which have the following features:

- They are universal approximators.
- They have no local minima.
- They have moderate complexity in terms of computation and memory.

In other words, we want to build a nonlinear adaptive filter that possesses the ability of modeling any continuous input–output mapping $y = f(\mathbf{u})$ and obeys the following sequential learning rule:

$$f_i = f_{i-1} + \mathbf{Gain}(i)e(i) \tag{1.23}$$

where f_i denotes the estimate of the mapping at time i and $\mathbf{Gain}(i)$ is a function in general. This sequential learning, which was first studied by Goodwin and Sin [1984] for linear filters, is attractive in practice because the current

estimate consists of two additive parts, namely, the previous estimate and a correction term proportional to the prediction error on new data. This unique incremental nature distinguishes our methods from all the others. Although equation (1.23) seems simple, the algorithm can in fact be motivated by many different objective functions. Also, depending on the precise meanings of **Gain**(i) and $e(i)$, the algorithm can take many different forms. We explore this in detail in the subsequent chapters. This amazing feature is achieved with the underlying linear structure of the reproducing kernel Hilbert space where the algorithms exist, as is discussed next.

1.4 REPRODUCING KERNEL HILBERT SPACES

A *pre-Hilbert space* is an *inner product space* that has an *orthonormal basis* $\{\mathbf{x}_k\}_{k=1}^{\infty}$. Let $\mathbb{H}$ be the largest and most inclusive space of vectors for which the infinite set $\{\mathbf{x}_k\}_{k=1}^{\infty}$ is a basis. Then, vectors not necessarily lying in the original inner product space represented in the form

$$\mathbf{x} = \sum_{k=1}^{\infty} a_k \mathbf{x}_k$$

are said to be spanned by the basis $\{\mathbf{x}_k\}_{k=1}^{\infty}$; the a_k are the coefficients of the representation. Define the new vector

$$\mathbf{y}_n = \sum_{k=1}^{n} a_k \mathbf{x}_k$$

Another vector $\mathbf{y}_m$ may be similarly defined. For $n > m$, we may express the squared Euclidean distance between the vectors $\mathbf{y}_n$ and $\mathbf{y}_m$ as

$$\begin{aligned} \|\mathbf{y}_n - \mathbf{y}_m\|^2 &= \left\| \sum_{k=1}^{n} a_k \mathbf{x}_k - \sum_{k=1}^{m} a_k \mathbf{x}_k \right\|^2 \\ &= \left\| \sum_{k=m+1}^{n} a_k \mathbf{x}_k \right\|^2 \\ &= \sum_{k=m+1}^{n} a_k^2 \end{aligned}$$

where, in the last line, we invoked the orthonormality condition. Therefore, to make the definition of $\mathbf{x}$ meaningful, we need the following to hold:

1. $\sum_{k=m+1}^{n} a_k^2 \to 0$ as both $n, m \to \infty$.
2. $\sum_{k=1}^{m} a_k^2 < \infty$.

In other words, a sequence of vectors $\{\mathbf{y}_k\}_{k=1}^{\infty}$ so defined is a *Cauchy sequence*. Consequently, a vector $\mathbf{x}$ can be expanded on the basis $\{\mathbf{x}_k\}_{k=1}^{\infty}$ if, and only if, $\mathbf{x}$ is a linear combination of the basis vectors and the associated coefficients $\{a_k\}_{k=1}^{\infty}$ are square summable. From this discussion, it is apparent that the space $\mathbb{H}$ is more "complete" than the starting inner product space. We may therefore make the following important statement:

> *An inner product space $\mathbb{H}$ is complete if every Cauchy sequence of vectors taken from the space $\mathbb{H}$ converges to a limit in $\mathbb{H}$; a complete inner product space is called a* Hilbert space.

A *Mercer kernel* [Aronszajn, 1950] is a continuous, symmetric, positive-definite function $\kappa : \mathbb{U} \times \mathbb{U} \rightarrow \mathbb{R}$. $\mathbb{U}$ is the input domain, a subset of $\mathbb{R}^L$. The commonly used kernels include the Gaussian kernel [equation (1.24)] and the polynomial kernel [equation (1.25)]:

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp\left(-a\|\mathbf{u} - \mathbf{u}'\|^2\right) \tag{1.24}$$

$$\kappa(\mathbf{u}, \mathbf{u}') = \left(\mathbf{u}^T\mathbf{u}' + 1\right)^p \tag{1.25}$$

Let $\mathbb{H}$ be any vector space of all real-valued functions of $\mathbf{u}$ that are generated by the kernel $\kappa(\mathbf{u}, \cdot)$. Suppose now two functions $h(\cdot)$ and $g(\cdot)$ are picked from the space $\mathbb{H}$ that are respectively represented by

$$h = \sum_{i=1}^{l} a_i \kappa(\mathbf{c}_i, \cdot)$$

and

$$g = \sum_{j=1}^{m} b_j \kappa(\tilde{\mathbf{c}}_j, \cdot)$$

where the a_i and the b_j are expansion coefficients and both $\mathbf{c}_i$ and $\tilde{\mathbf{c}}_j \in \mathbb{U}$ for all i and j. The *bilinear form* defined as

$$\langle h, g \rangle = \sum_{i=1}^{l} \sum_{j=1}^{m} a_i \kappa(\mathbf{c}_i, \tilde{\mathbf{c}}_j) b_j$$

satisfies the following properties:

1. Symmetry

$$\langle h, g \rangle = \langle g, h \rangle$$

2. Scaling and distributive property

$$\langle (cf + dg), h \rangle = c\langle f, h \rangle + d\langle g, h \rangle$$

3. Squared norm

$$\|f\|^2 = \langle f, f \rangle \geq 0$$

By virtue of these facts, the bilinear term $<h, g>$ is indeed an inner product. There is one additional property that follows directly. Specifically, setting $g(\cdot) = \kappa(\mathbf{u}, \cdot)$, we obtain

$$\begin{aligned} \langle h, \kappa(\mathbf{u}, \cdot) \rangle &= \sum_{i=1}^{l} a_i \kappa(\mathbf{c}_i, \mathbf{u}) \\ &= h(\mathbf{u}) \end{aligned}$$

This property is known as the *reproducing property*. The kernel $\kappa(\mathbf{u}, \mathbf{u}')$, which represents a function of the two vectors $\mathbf{u}$, $\mathbf{u}' \in \mathbb{U}$, is called a reproducing kernel of the vector space $\mathbb{H}$ if it satisfies the following two conditions:

1. For every $\mathbf{u} \in \mathbb{U}$, $\kappa(\mathbf{u}, \mathbf{u}')$ as a function of the vector $\mathbf{u}'$ belongs to $\mathbb{H}$.
2. It satisfies the reproducing property.

These two conditions are indeed satisfied by the Mercer kernel, thereby endowing it with the designation "reproducing kernel." If the inner product space $\mathbb{H}$, in which the reproducing kernel space is defined, is also complete, then it is called a reproducing kernel Hilbert space (RKHS).

The analytic power of RKHS is expressed in an important theorem called the Mercer theorem. The Mercer theorem [Aronszajn, 1950, Burges, 1998] states that any reproducing kernel $\kappa(\mathbf{u}, \mathbf{u}')$ can be expanded as follows:

$$\kappa(\mathbf{u}, \mathbf{u}') = \sum_{i=1}^{\infty} \varsigma_i \phi_i(\mathbf{u}) \phi_i(\mathbf{u}') \tag{1.26}$$

where ς_i and ϕ_i are the eigenvalues and the eigenfunctions, respectively. The eigenvalues are non-negative. Therefore, a mapping $\boldsymbol{\varphi}$ can be constructed as

$$\begin{aligned} &\boldsymbol{\varphi} : \mathbb{U} \rightarrow \mathbb{F} \\ &\boldsymbol{\varphi}(\mathbf{u}) = \left[\sqrt{\varsigma_1} \phi_1(\mathbf{u}), \sqrt{\varsigma_2} \phi_2(\mathbf{u}), \ldots \right] \end{aligned} \tag{1.27}$$

By construction, the dimensionality of $\mathbb{F}$ is determined by the number of strictly positive eigenvalues, which are infinite in the Gaussian kernel case.

In the machine learning literature, $\boldsymbol{\varphi}$ is usually treated as the feature mapping and $\boldsymbol{\varphi}(\mathbf{u})$ is the transformed feature vector lying in the feature space $\mathbb{F}$ (which is an inner product space) (Figure 1.4). By doing so, an important implication is

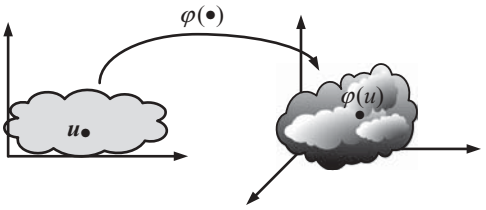


Figure 1.4. Nonlinear map $\boldsymbol{\varphi}(\cdot)$ from the input space to the feature space.

$$\boldsymbol{\varphi}(\mathbf{u})^T \boldsymbol{\varphi}(\mathbf{u}') = \kappa(\mathbf{u}, \mathbf{u}') \tag{1.28}$$

It is easy to check that $\mathbb{F}$ is essentially the same as the RKHS induced by the kernel by identifying $\boldsymbol{\varphi}(\mathbf{u}) = \kappa(\mathbf{u}, \cdot)$, which are the bases of the two spaces, respectively. By slightly abusing the notation, we do not distinguish $\mathbb{F}$ and $\mathbb{H}$ in this book if no confusion is involved.

A concrete example helps here. Let [Cherkassky and Mulier, 1998]

$$\kappa(\mathbf{u}, \mathbf{c}) = \left(1 + \mathbf{u}^T \mathbf{c}\right)^2 \tag{1.29}$$

with $\mathbf{u} = [u_1, u_2]^T$ and $\mathbf{c} = [c_1, c_2]^T$. By expressing the polynomial kernel in terms of monomials of various orders, we have

$$\kappa(\mathbf{u}, \mathbf{c}) = 1 + u_1^2 c_1^2 + 2u_1 u_2 c_1 c_2 + u_2^2 c_2^2 + 2u_1 c_1 + 2u_2 c_2$$

Therefore, the image of the input vector $\mathbf{u}$ in the feature space may be written as

$$\boldsymbol{\varphi}(\mathbf{u}) = \left[1, u_1^2, \sqrt{2}u_1 u_2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2\right]^T$$

And similarly we have

$$\boldsymbol{\varphi}(\mathbf{c}) = \left[1, c_1^2, \sqrt{2}c_1 c_2, c_2^2, \sqrt{2}c_1, \sqrt{2}c_2\right]^T$$

It is easy to verify that

$$\boldsymbol{\varphi}(\mathbf{u})^T \boldsymbol{\varphi}(\mathbf{c}) = \kappa(\mathbf{u}, \mathbf{c})$$

However, it is hard in general to express $\boldsymbol{\varphi}$ explicitly even for simple polynomial kernels, because the dimensionality of $\boldsymbol{\varphi}$ scales with $O(L^p)$, where L is the dimension of input vectors and p is the order of the polynomial kernel.

1.5 KERNEL ADAPTIVE FILTERS

The *kernel method* is a powerful nonparametric modeling tool. The main idea can be summarized as follows: Transform the input data into a high-dimensional feature space via a reproducing kernel such that the inner product operation in the feature space can be computed efficiently through the kernel evaluations [equation (1.28)]. Then, appropriate linear methods are subsequently applied on the transformed data. As long as an algorithm can be formulated in terms of inner products (or equivalent kernel evaluation), there is no need to perform computations in the high-dimensional feature space. Even though this methodology is called the "kernel trick," we have to point out that the underlying reproducing kernel Hilbert space plays a central role in providing linearity, convexity, and universal approximation capability. Successful examples of this methodology include support vector machines, kernel principal component analysis, and Fisher discriminant analysis.[8]

We start with an example to show why projecting the input into a feature space helps in learning. Consider the target function of a two-dimensional input $\mathbf{u} = [u_1, u_2]^T$.

$$f(u_1, u_2) = a_1 u_1 + a_2 u_2 + a_3 u_1^2 + a_4 u_2^2 \tag{1.30}$$

where a_1, a_2, a_3, and a_4 are some constant coefficients. Apparently, a linear system trying to approximate f by a linear combination of u_1 and u_2 could not exactly model it as written. However, by using the kernel [equation (1.29)] and its mapping $\boldsymbol{\varphi}$, we have a new representation of the input

$$(u_1, u_2) \overset{\varphi}{\mapsto} (x_1, x_2, x_3, x_4, x_5, x_6) = \left(1, u_1^2, \sqrt{2}u_1u_2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2\right)$$

Now, f can be represented by a linear system of $(x_1, x_2, x_3, x_4, x_5, x_6)$

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = 0 \cdot x_1 + a_3 x_2 + 0 \cdot x_3 + a_4 x_4 + \frac{a_1}{\sqrt{2}} x_5 + \frac{a_2}{\sqrt{2}} x_6$$

The fact that mapping the input into a feature space can simplify the learning task has been well known for a long time in machine learning as exemplified by polynomial regression[9] and Volterra series. The problem is really how to construct this mapping. One may be tempted to add as many features as possible because it is more likely the target functions can be represented using a standard learning algorithm in high-dimensional feature spaces, but this may run into the danger of *overfitting*. Overfitting is a phenomenon where a good fit to the training data is achieved, but the overlearned system performs badly when making test predictions. The difficulties with high-dimensional feature spaces are mainly as follows:

1. The computational complexity explodes with the dimensionality.
2. The generalization performance degrades as the dimensionality increases.

Two approaches can be used to overcome these problems. One is called *feature selection* where only useful features are selected and other features are pruned so that the dimensionality of the feature space is constrained. In our previous example, apparently x_1 and x_3 are two *redundant features* that should be pruned. The other workaround is the kernel method. Because the features are only implicitly constructed and we do not need to work directly in the feature space, the explosion of computational complexity is avoided. The overfitting problem is taken care of by the use of regularization. These properties will become clear when we develop the kernel adaptive filters in the subsequent chapters.

It has been proved [Steinwart, 2001] that in the case of the Gaussian kernel, for any continuous input–output mapping $f : \mathbb{U} \to \mathbb{R}$ and any $\varsigma > 0$, there exist parameters $\{\mathbf{c}_i\}_{i=1}^m$ in $\mathbb{U}$ and real numbers $\{a_i\}_{i=1}^m$ such that

$$\left\| f - \sum_{i=1}^{m} a_i \kappa(\cdot, \mathbf{c}_i) \right\|_2 < \varsigma \tag{1.31}$$

If we denote a vector $\boldsymbol{\omega}$ in $\mathbb{F}$ as

$$\boldsymbol{\omega} = \sum_{i=1}^{m} a_i \boldsymbol{\varphi}(\mathbf{c}_i)$$

then by equations (1.28) and (1.31), we have

$$\left\| f - \boldsymbol{\omega}^T \boldsymbol{\varphi} \right\|_2 < \varsigma$$

This equation implies that the linear model in $\mathbb{F}$ has the *universal approximation property*. Clearly, this property is established from the viewpoint of strict function approximation.

Furthermore, if our problem is to minimize a regularized cost function over a finite data set $\{(\mathbf{u}(i), d(i))\}_{i=1}^N$, then we write

$$\min_f J(f) = \sum_{i=1}^{N} (d(i) - f(\mathbf{u}(i)))^2 + \lambda \|f\|_2^2$$

It has been shown that the optimal solution can be expressed as

$$f = \sum_{i=1}^{N} a_i \kappa(\cdot, \mathbf{u}(i))$$

Table 1.1. Comparison of different nonlinear adaptive filters.

Algorithms	Modeling capacity	Convexity	Complexity
Linear adaptive filters	Linear only	Yes	Very simple
Hammerstein, Wiener models	Limited nonlinearity	No	Simple
Volterra, Wiener series	Universal	Yes	Very high
Time-lagged neural networks	Universal	No	Modest
Recurrent neural networks	Universal	No	High
Kernel adaptive filters	Universal	Yes	Modest
Recursive Bayesian estimation	Universal	No	Very high

for suitable a_i. This result is called the *representer theorem* [Schölkopf et al., 2001]. In other words, although we did consider functions that were expansions in terms of arbitrary points $\mathbf{c}_i$ [see equation (1.31)], it turns out that we can always express the solution in terms of the training points $\mathbf{u}(i)$ only. Hence, the optimization problem over an arbitrarily large number of variables is transformed into one over N variables, where N is the number of training points.

Recently, it also has been shown that Volterra series and Wiener series can be treated just as a special case of a kernel regression framework [Franz and Schölkopf, 2006]. By formulating the Volterra and Wiener series as a linear regression in RKHS, the complexity is now independent of the input dimensionality and the order of nonlinearity.

Based on these advantages and arguments, our strategy is clear: to formulate the classic adaptive filters in RKHS such that we are iteratively solving a convex least-squares problem there. As long as we can formulate these algorithms in terms of inner products, we obtain nonlinear adaptive filters that have the universal approximation property and convexity at the same time. Convexity is an important feature that prevents the algorithms from being stuck in local minima. (See Table 1.1.)

In recent years, many efforts of "kernelizing" adaptive filters have been published in the literature. Frieb and Harrison [1999] first used this idea to derive the kernel ADALINE, which is formulated as a deterministic gradient method based on all the training data (not online). Then, Kivinen et al. [2004] proposed an algorithm called NORMA by directly differentiating a regularized functional cost to get the stochastic gradient. Although the derivation involves advanced mathematics, the results are actually equivalent to a kernel version of the leaky least-mean-square algorithm. At almost the same time, Engel et al. [2004] studied the case of kernel recursive least squares by using the matrix inversion lemma. Later on, Liu et al. [2008] investigated the kernel least-mean-square algorithm and pointed out that the algorithm possesses a self-regularization property. Kernel affine projection algorithms were studied from different perspectives by Liu and Príncipe [2008b], Slavakis and Theodoridis [2008], and Richard et al. [2009]. More recently, the extended

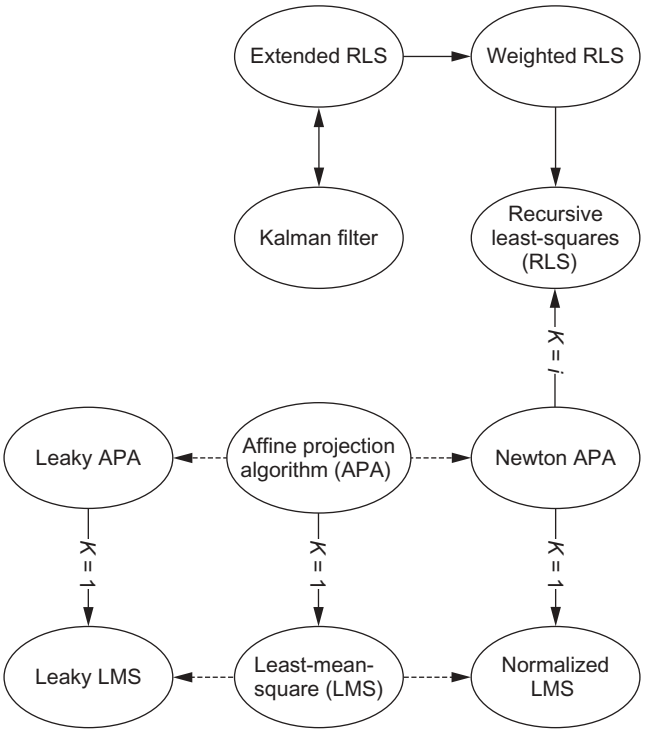


Figure 1.5. Relation among different adaptive filtering algorithms.

kernel recursive least-squares algorithm was presented in Liu et al. [2009], who studied the general state estimation problem in RKHS. After a decade of efforts from many researchers, kernel adaptive filtering is rapidly evolving into an important field of signal processing. This book serves to address previous works and incorporate them in a unifying framework (see Figure 1.5). We present, in detail, the kernel least-mean-square algorithm, the kernel affine projection algorithms, the kernel recursive least-squares algorithm, and the extended kernel recursive least-squares algorithm. Relations among these algorithms, which are illustrated in Figure 1.5, will become clear when we present them in the subsequent chapters.

Kernel adaptive filters provide a generalization of linear adaptive filters because these become a special case of the former when expressed in the dual space. Kernel adaptive filters exhibit a growing memory structure embedded in the filter weights. They naturally create a growing radial-basis function network, learning the network topology and adapting the free parameters directly from data at the same time. The learning rule is a beautiful combination of the error-correction and memory-based learning, and potentially it will have a deep impact on our understanding about the essence of kernel learning theory.

Historically, most kernel methods use block adaptation and are computationally expensive using a large Gram matrix of dimensionality given by the number of data points; therefore, the efficient online algorithms provide the useful flexibility for trading off performance with complexity. And in nonstationary environments, the tracking ability of online algorithms provides an extra advantage.

The combination of sequential learning and memory-based learning requires and at the same time enables, the network to select informative training examples instead of treating all examples equally. Empirical evidence shows that selecting informative examples can reduce drastically the training time and produce much more compact networks with equivalent accuracy. Therefore, in the case of a large and redundant data set, performing kernel online learning algorithms provides a big edge over batch mode methods in terms of efficiency.

The widely used active data selection methods for kernel adaptive filters include the novelty criterion [Platt, 1991] and approximate linear dependency test [Engel et al., 2004]. Both are based on heuristic distance functions while we present a principled and unifying approach. Our criterion is based on a subjective information measure called "surprise". It quantifies how informative the candidate exemplar is relative to the knowledge of the learning system. It turns out that the approximate linear dependency test is a special case and that the novelty criterion is some approximation in this information theoretic framework.

1.6 SUMMARIZING REMARKS

To put the introductory material covered in this chapter into a historical context, it is noteworthy that in a classic paper on the separability of patterns published in 1965, Cover proved that, given a nonlinearly separable pattern-classification problem, there is, in general, a practical benefit to be gained in mapping the input (data) space into a hidden (feature) space of high enough dimensionality. Basically, a nonlinearly separable pattern-classification problem is transformed into a linearly separable one, provided that the following two conditions are satisfied:

1. The transformation from the input space into the feature space is nonlinear.
2. The dimensionality of the feature space is high enough.

Inspired by Cover's insightful ideas just summarized, the following statement was made in Chapter 7 of the first edition of *Neural Networks* [Haykin, 1994, p. 242]:

> In a similar fashion, we may use a nonlinear mapping to transform a difficult nonlinear filtering problem into an easier one that involves linear filtering.

Unfortunately, the extension of Cover's ideas to nonlinear adaptive filtering problems is more difficult because we now have to account for time in an online manner.

Much has been written on the theory and design of nonlinear adaptive filters, going back to an early paper from Gabor et al. [1960]. However, an elegant, unified theory for the algorithmic implementation of nonlinear adaptive filters, which builds on the well-established linear adaptive filters [Widrow and Stearns, 1985, Haykin, 1996], has been lacking. In light of the introductory material presented in this chapter, followed by the detailed expositions presented in the rest of the book on different procedures of implementing kernel adaptive filters, it can be justifiably said that at long last we now have the elegant, unified theory that has been lacking in the literature for much too long.

The new theory presented in this book is elegant because it builds on the Mercer theorem, which is basic to the well-established kernel methods. Moreover, the theory is unified in that it also exploits all the powerful linear adaptive filtering algorithms, namely, the LMS and RLS algorithms and their respective modified versions. Simply stated, the new theory of kernel adaptive filters brings together two different subjects under a single umbrella:

- The Mercer kernel theory, which, in practical terms, manifests itself in the form of memory
- Linear adaptive filter theory, through which the need for adaptation is taken care of

Most importantly, the new theory is developed in a coherent fashion.

ENDNOTES

1. **Model Selection Criteria.** There are mainly three tools for model selection: Akaike information criterion (AIC), Bayesian information criterion (BIC), and minimum description length (MDL). *Akaike's information criterion* was developed by Hirotsugu Akaike under the name of "Akaike information criterion" in 1971 and proposed in Akaike [1974]. AIC is a measure of the goodness of fit of an estimated statistical model, which is defined as

$$\mathrm{AIC} = 2k - 2\ln(L_{\max}) \tag{1.32}$$

where k is the number of the free parameters in the model and $L_{\max}$ is the maximized value of the likelihood function for the model. Given a data set, several competing models may be ranked according to their AIC; the one with the lowest AIC is the best. If the model errors are normally and independently distributed, then AIC may simplify to

$$\mathrm{AIC} = 2k + 2N\ln(\mathrm{MSE}) \tag{1.33}$$

where N is the number of data points and MSE is the mean square error of the data by using the model. Equation (1.33) consists of two terms: measure of the goodness of fit and penalty of the model complexity. In this sense, the AIC methodology attempts to find the model that best explains the data with a minimum of free parameters.

Bayesian information criterion was developed by Schwarz [1978]; it is closely related to the Akaike information criterion. The formula for BIC is

$$\text{BIC} = k \ln(N) - 2\ln(L_{\max}) \tag{1.34}$$

Under the assumption that the model errors are Gaussian distributed, this equation becomes

$$\text{BIC} = k \ln(N) + N \ln(\text{MSE}) \tag{1.35}$$

The *Minimum description length* was introduced by Rissanen [1978] and systematically studied by Grünwald [2007]. The MDL principle is a formalization of Occam's Razor, in which the best hypothesis for a given set of data is the one that leads to the largest compression of the data. The ideal MDL approach requires the estimation of the Kolmogorov complexity, which is uncomputable in general. However, nonideal, practical versions of MDL are widely used in the machine learning community; see for example Grünwald [2007], and Haykin [2009].

2. **Growing and Pruning Neural Networks.** The *adaptive resonance theory* architecture [Grossberg, 1987, Carpenter and Grossberg, 1987] is one of the first growing neural networks. It is a biologically inspired approach rather than a computational design. However, it is often inefficient and the convergence of the iterative processing is not guaranteed.

   The *cascade-correlation learning architecture* [Fahlman and Lebiere, 1990] is another example of the network-growing approach. The procedure begins with a minimal network that has some input and one or more output nodes as indicated by input–output considerations, but no hidden nodes. The hidden neurons are added to the network one by one, thereby obtaining a multilayer structure.

   Another network-growing approach is described in Lee et al. [1990], where a third level of computation is added to the forward pass (function-level adaptation) and the backward pass (parameter-level adaptation). In this third level of computation, the structure of the network is adapted by changing the number of neurons and the structural relationship among neurons in the network. This method of network growing is computationally intensive.

   Platt [1991] proposed a more feasible design called *resource allocating networks*, where the structure of a neural network was dynamically altered to optimize resource allocation. Since then, many researchers have proposed methods of both growing and pruning radial-basis function networks reported in Cheng and Lin [1994], Karayiannis and Mi [1997], and Huang et al. [2005].

   Martinetz and Schulten [1991] started a new strand of research into growing networks by inventing the "neural gas" models, where the topologies were not predetermined and connections between nodes were added as needed. These "neural gas" models are self-organizing systems that use Hebb-like learning rule to learn the distribution of input data.

Two main approaches to network pruning are 1) regularization, of which notable examples are *weight decay* [Hinton and Sejnowski, 1986], *weight elimination* [Weigend et al., 1990], and *approximate smoother* [Moody and Rögnvaldsson, 1997]; and 2) systematic deletion, which includes the *optimal brain damage* procedure [LeCun et al., 1990] and the *optimal brain surgeon* procedure [Hassibi and Stork, 1992].

3. **Sparsification.** In kernel methods and Gaussian process modeling, the complexity is usually directly proportional to the number of training data either at linear scale, quadratic scale, or even cubic scale. Sparsification is a process of selecting only an "important" subset of the training data to train the model, and by so doing the complexity of the algorithm can be reduced greatly. There are two main approaches. One is by *elimination*, as in support vector machines [Vapnik, 1995], regularization networks [Evgeniou et al., 2000], relevance vector machines [Tipping, 2001], and least-squares support vector machines [Suykens et al., 2000]. These algorithms start by considering all training samples as potential centers. And part of the samples are eliminated by solving the optimization problem wherein the associated coefficients become zero. These algorithms are usually computationally expensive, scaling with $O(N^3)$ in time and up to $O(N^2)$ in space, where N is the number of training examples.

   The other approach is by *construction*. Here, the algorithm starts with an empty network and gradually adds centers at each step of the construction process. Because finding the best subset is a combinational optimization problem, these algorithms usually employ various greedy selection strategies, in which at each step the sample that maximizes the amount of some fitness criterion is selected [Seeger and Williams, 2003, Smola and Bartlett, 2001, Quinonero-Candela and Rasmussen, 2005]. Still, the complexity of these algorithms ranges from $O(M^2N)$ to $O(MN^2)$, where $M(<<N)$ is the size of the selected subset (or the number of basis functions). It usually requires multiple passes of the whole training data. If computer memory cannot hold the whole training data, then disk-read operations would slow down the learning speed significantly.

4. **Active learning.** Active learning has been studied under the names of *optimal experiment design* [Lindley, 1956, Fedorov, 1972], *sequential decision making* [El-Gamal, 1991], *query learning* [Campbell et al., 2000], and *selective sampling* [Lindenbaum, 1999] in such diversified fields as economics theory, statistics, and machine learning. The uses of active learning in neural networks are reported in MacKay [1992a], Fukumizu [1996], and Tong and Koller [2000]. Active learning in sequential methods has been studied in Platt [1991], Engel et al. [2004], and Csato and Opper [2002] for regression problems and in Bordes et al. [2005] and Glasmachers [2008] for classification problems.

5. **Linear Adaptive Filters.** The earliest work on adaptive filters may be traced back to the late 1950s, during which time many researchers were working independently on different applications of such filters. As a result, the LMS emerged as a simple and effective algorithm for the operation of adaptive transversal filters. The LMS algorithm was devised by Widrow and Hoff in 1959 in their study of an adaptive linear element, which is commonly referred to in the literature as the *ADALINE* [Widrow and Hoff, 1960].

   Another important algorithm in adaptive filtering theory is the RLS algorithm. The original paper on the standard RLS algorithm is that of Plackett [1950], although many other researchers are believed to have derived and rederived various versions

of the RLS algorithm. In 1974, Godard first used Kalman filter theory successfully to solve adaptive filtering problems, which is known in the literature as the *Godard algorithm*. Then, Sayed and Kailath [1994] established an exact relationship between the RLS algorithm and Kalman filter theory, thereby laying the groundwork for how to exploit the vast literature on Kalman filters for solving linear adaptive filtering problems.

The EX-RLS algorithm was first presented in Haykin et al. [1997]. It is derived as an improvement over the RLS algorithm in terms of tracking ability in nonstationary signal processing. The algorithm is a special case of the *Kalman filter* while derived from the perspective of adaptive signal processing. The Kalman filter is used in a wide range of engineering applications from radar to navigation, and it is an important topic in control theory and control systems engineering. The filter is named after Rudolf E. Kalman based on his seminal papers [Kalman, 1960, Kalman and Bucy, 1961], although Thorvald Nicolai Thiele and Peter Swerling are found to have developed a similar algorithm earlier [Lauritzen, 1981]. A highlight application of the Kalman filter is perhaps its incorporation in the Apollo navigation computer.

Two excellent textbooks for linear adaptive filtering were authored by Haykin [2002] and Sayed [2003].

6. **Nonlinear Adaptive Filters.** The idea of nonlinear adaptive filter was proposed by Gabor in 1954, who was one of the early pioneers of communications theory. Hammerstein and Wiener models are discussed in Wiener [1958], Billings and Fakhouri [1982], and Marmarelis [1993]. Volterra [1887] and later Gabor [1968] studied the use of Volterra series for nonlinear adaptive filtering. The Wiener series was proposed and studied in Wiener [1958] and Barrett [1963] as an improvement over the Volterra series. For the use of time-lagged multilayer perceptrons, radial-basis function networks, and recurrent neural networks for nonlinear adaptive filtering, see Lang and Hinton [1988], Wan [1990], Príncipe et al. [1992], and Haykin [1998].

7. **Volterra Series.** The Volterra series is named after the Spanish mathematician Vito Volterra, who first introduced the notion in 1887 [Volterra, 1887]. The first major application of Volterra's work to nonlinear circuit analysis was done by the mathematician Norbert Wiener at the Massachusetts Institute of Technology, who used them in a general way to analyze several problems including the spectrum of an FM system with a Gaussian noise input [Wiener, 1958].

   The Volterra series is a model for a nonlinear, time-invariant system with memory. It is similar to the Taylor series in terms of nonlinear modeling but differs from the Taylor series in its ability to capture "memory" effects. The Taylor series can be used to approximate the nonlinear response of a system to a given input:

$$y(t) = \sum_{n=0}^{\infty} a_n [x(t)]^n$$

   where the output $y(t)$ depends strictly on the input $x(t)$ at that particular time. In the Volterra series, the output of the nonlinear system depends on all the input that has been applied to the system in the past. This provides the ability to capture the "memory" effect of devices such as capacitors and inductors. A linear, causal system with memory can be described by the convolution representation:

$$y(t) = \int_{-\infty}^{\infty} h(\tau) x(t-\tau) d\tau$$

where $y(t)$ is the output, $x(t)$ is the input, and $h(t)$ is the impulse response of the system. The Volterra series combines these two techniques to describe a nonlinear, dynamic, time-invariant system in the following way:

$$\begin{aligned} y(t) &= \sum_{n=0}^{\infty} \frac{1}{n!} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} k_n(\tau_1, \tau_2, \ldots, \tau_n) x(t-\tau_1) x(t-\tau_2) \cdots x(t-\tau_n) d\tau_1 d\tau_2 \cdots d\tau_n \\ &= k_0 \\ &\quad + \frac{1}{1!} \int_{-\infty}^{\infty} k_1(\tau_1) x(t-\tau_1) d\tau_1 \\ &\quad + \frac{1}{2!} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} k_2(\tau_1, \tau_2) x(t-\tau_1) x(t-\tau_2) d\tau_1 d\tau_2 \\ &\quad + \frac{1}{3!} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} k_3(\tau_1, \tau_2, \tau_3) x(t-\tau_1) x(t-\tau_2) x(t-\tau_3) d\tau_1 d\tau_2 d\tau_3 \\ &\quad + \cdots \end{aligned} \tag{1.36}$$

where the $k_n(\tau_1, \tau_2, \ldots, \tau_n)$ are called the *Volterra kernels* of the system. For $n = 1$, $k_1(\tau_1)$ is the conventional impulse response like in the linear system; for $n > 1$, k_n are regarded as "higher order impulse responses."

The determination of the Volterra kernels are generally complicated. Common methods include the *harmonic input method*, *direct expansion method*, and *powers of transfer function method*. The Volterra series is successful to model systems that exhibit "weak nonlinearity." If the system to be modeled is "strongly nonlinear," then the Volterra series either takes a long time to converge or often diverges. For more details, read Cherry [1994] and Schetzen [2006].

8. **Kernel Methods.** The idea of using kernel functions as inner products in a feature space was introduced into machine learning by the work of Aizerman et al. [1964] on the method of potential functions. Then, Boser et al. [1992] combined the idea with large margin classifiers, leading to the birth of support vector machines and the popularity of kernel in the machine learning community. Schölkopf et al. [1998] derived the first unsupervised learning algorithm in reproducing kernel Hilbert space by introducing the kernel principal components analysis. The description of kernel Fisher discriminant analysis can be found in Mika et al. [1999]. The use of kernels for function approximation dates back to Aronszajn [1950]. Then, Wahba [1990] systematically studied reproducing kernels in approximation and regularization theory. At the same time, Poggio and Girosi [1990] used reproducing kernels in the development of regularization networks. Excellent tutorial books on kernel methods include Cristianini and Shawe-Taylor [2000], Shawe-Taylor and Cristianini [2004], and Schölkopf and Smola [2002]. More advanced reading on support vector machines includes Vapnik's books on statistical learning theory [Vapnik, 1995, 1998].
9. **Polynomial Regression.** The first use of polynomial regression was published by Gergonne [Stigler, 1974]. The polynomial regression is a simple generalization of the linear regression model. The goal of regression analysis is to model the expected value of a dependent variable y in terms of the value of an independent variable x. In simple linear regression, the model is

$$y = a_0 + a_1 x + \varepsilon$$

where ε is random noise with zero mean. Similarly a quadratic model is

$$y = a_0 + a_1 x + a_2 x^2 + \varepsilon$$

In general, we can use an nth-order polynomial to model the mapping between two scalar variables y and x:

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n + \varepsilon$$

The advantage of the method is it is easy to estimate the linear coefficients a_0, a_1, …, a_n using the least-squares analysis. Polynomial regression belongs to a more general function approximation framework using basis functions, which include splines, radial-basis functions, and wavelets. The drawbacks of polynomial bases are 1) they are correlated and 2) they are nonlocal. These drawbacks lead to practical difficulties in terms of interpretation and stability.

2

KERNEL LEAST-MEAN-SQUARE ALGORITHM

The great appeal of developing filters in reproducing kernel Hilbert spaces (RKHS) is to use the linear structure of this space to implement well-established linear adaptive algorithms and to obtain nonlinear filters in the input space. When compared with neural networks, this alternative design approach leads to universal approximation capabilities, convex optimization (i.e., no local minima), and computational complexity that is still reasonable. It holds a unique position by bridging two important areas of adaptive filtering and neural networks. The bottleneck of the RKHS approach to nonlinear filter design is the need for regularization, the need to select the kernel function, and the need to curtail the growth of the filter structure.

In particular, in this chapter, we study the kernel least-mean-square (KLMS) algorithm, which is the simplest among the family of the kernel adaptive filters in Figure 1.5. The linear least-mean-square (LMS) algorithm will be directly mapped into RKHS with an emphasis on the general methodology to formulate linear filters and gradient descent algorithms in terms of inner products that can immediately take advantage of the reproducing property of RKHS and be directly implemented by kernel evaluations. This is a critical step in the overall design; otherwise, the RKHS methodology loses its edge because of the insurmountable computational complexity of operating with an infinite number of parameters.

Another important aspect for understanding is to show how the nonlinear filter is incrementally constructed during adaptation. The KLMS algorithm naturally creates a growing radial-basis function network, learns network

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

topology, and adapts free parameters directly from training data. Kernel filtering is a memory-intensive operation just like other kernel methods. However, kernel filtering is online and the filter output is incrementally constructed by using previous samples and prediction errors.

The material presented in this chapter also includes a detailed analysis of the KLMS self-regularization property. Thanks to its gradient descent nature, KLMS does not need extra solution norm constraint in contrast to most kernel methods. This simplifies even further the implementation and provides a practical nonlinear filter design.

2.1 LEAST-MEAN-SQUARE ALGORITHM

Suppose the goal is to learn a continuous input–output mapping $f : \mathbb{U} \rightarrow \mathbb{R}$ based on a sequence of input–output examples $\{\mathbf{u}(1), d(1)\}$, $\{\mathbf{u}(2), d(2)\}$, …, $\{\mathbf{u}(N), d(N)\}$. $\mathbb{U}$ is the input domain and is assumed as a subspace of $\mathbb{R}^L$. The output is assumed to be one-dimensional, but it is straightforward to generalize the discussion to multidimensional output. N is the size of training data; the problem of sequential learning with infinite training data will be addressed later.

The LMS algorithm assumes a linear model and uses the following procedure:

$$\begin{aligned} \mathbf{w}(0) &= 0 \\ e(i) &= d(i) - \mathbf{w}(i-1)^T \mathbf{u}(i) \\ \mathbf{w}(i) &= \mathbf{w}(i-1) + \eta e(i)\mathbf{u}(i) \end{aligned} \tag{2.1}$$

to find approximately the optimal weight vector $\mathbf{w}^o$, which minimizes the empirical risk:

$$J(\mathbf{w}) = \sum_{i=1}^{N} \left(d(i) - \mathbf{w}^T \mathbf{u}(i)\right)^2$$

In equation (2.1), $e(i)$ is called the prediction error, η is the step-size parameter, and $\mathbf{w}(i)$ is the estimate of the optimal weight at iteration i. LMS can be derived by using the *instantaneous gradient*. The gradient of the cost function with respect to $\mathbf{w}$ is

$$\nabla_{\mathbf{w}} J = -2 \sum_{i=1}^{N} \mathbf{u}(i)\left(d(i) - \mathbf{w}^T \mathbf{u}(i)\right) \tag{2.2}$$

and the instantaneous gradient at time i is

$$\widehat{\nabla_{\mathbf{w}} J} = -\mathbf{u}(i)\left(d(i) - \mathbf{w}(i-1)^T \mathbf{u}(i)\right) \tag{2.3}$$

by dropping the summation and neglecting the constant 2 for simplicity. Finally, according to the method of steepest descent, we may formulate the LMS algorithm as follows:

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta \mathbf{u}(i)\left(d(i) - \mathbf{w}(i-1)^T \mathbf{u}(i)\right) \tag{2.4}$$

For this reason, the LMS algorithm is sometimes referred to as a "stochastic gradient algorithm." The LMS algorithm is summarized in Algorithm 1, which clearly illustrates the simplicity of the algorithm. For the *initialization* of the algorithm, it is customary to set the initial value of the weight vector equal to zero.

Algorithm 1. The least-mean-square algorithm

Initialization

$\mathbf{w}(0) = 0$, choose η

Computation

while $\{\mathbf{u}(i), d(i)\}$ available **do**
  $e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i)$
  $\mathbf{w}(i) = \mathbf{w}(i-1) + \eta e(i)\mathbf{u}(i)$
end while

At iteration i, given a test point $\mathbf{u}_*$, the output of the system is

$$f(\mathbf{u}_*) = \mathbf{u}_*^T \mathbf{w}(i)$$

2.1.1 Convergence Considerations of the LMS Algorithm

The first criterion for convergence of the LMS algorithm is *convergence in the mean*, which is described by

$$\mathbf{E}[\mathbf{w}(i)] \to \mathbf{w}^o \quad \text{as } i \to \infty, N \to \infty \tag{2.5}$$

However, this criterion is too weak to be of any practical value, because a sequence of zero-mean, but otherwise arbitrary random, vectors converges in this sense.

A more practical convergence criterion is *convergence in the mean square*, which is described by

$$\mathbf{E}\left[e(i)^2\right] \to \text{constant} \quad \text{as } i \to \infty, N \to \infty \tag{2.6}$$

Under the assumption that the step-size parameter η is sufficiently small, it is shown in [Haykin, 2002] that the LMS is convergent in the mean square provided that η satisfies the condition

$$0 < \eta < \frac{1}{\varsigma_{\max}} \tag{2.7}$$

where $\varsigma_{\max}$ is the *largest eigenvalue* of the correlation matrix $\mathbf{R}_{\mathbf{u}}$, defined by

$$\mathbf{R}_{\mathbf{u}} = \sum_{i=1}^{N} \mathbf{u}(i)\mathbf{u}(i)^T \tag{2.8}$$

In typical applications of the LMS algorithm, knowledge of $\varsigma_{\max}$ is not available. To overcome this difficulty, the *trace* of $\mathbf{R}_{\mathbf{u}}$ may be taken as a conservative estimate for $\varsigma_{\max}$. Therefore, we have the following conservative condition:

$$0 < \eta < \frac{1}{\mathrm{tr}[\mathbf{R}_{\mathbf{u}}]} \tag{2.9}$$

2.1.2 Misadjustment of the LMS Algorithm

Another important parameter of the LMS algorithm is called the *misadjustment*, which is formally defined as

$$\mathcal{M} = \frac{J(\infty) - J_{\min}}{J_{\min}} \tag{2.10}$$

where $J(\infty)$ is the limiting constant of the mean square error $\mathbf{E}[e(i)^2]$ as i goes to ∞ and $J_{\min}$ is the irreducible error power caused by model mismatch and/or noise in the observations. In other words, the misadjustment is defined as the ratio of the steady-state value of the excess mean square error to the minimum mean square error. Under the small step-size theory, we may also write

$$\mathcal{M} = \frac{\eta}{2} \sum_{i=1}^{L} \varsigma_i \tag{2.11}$$

which, by the eigen-decomposition theory, is equivalent to

$$\mathcal{M} = \frac{\eta}{2} \mathrm{tr}[\mathbf{R}_{\mathbf{u}}] \tag{2.12}$$

The misadjustment is a dimensionless parameter that provides a measure of how close the LMS algorithm is to optimality in the mean-square-error sense. The smaller the misadjustment is compared with unity, the more accurate is the adaptive filtering action being performed by the LMS algorithm. It

is customary to express misadjustment as a percentage. For example, a misadjustment of 10% means that the LMS algorithm produces a mean square error (after adaptation is completed) that is 10% greater than the minimum mean square error $J_{\min}$. Such performance is ordinarily considered to be satisfactory in practice.

2.1.3 Learning Curve

Learning curve is an informative way of examining the convergence behavior of the LMS algorithm or in general any adaptive filter. We will use the learning curve a great deal in our experiments to compare the performance of different adaptive filters. The learning curve is a plot of the mean square error (MSE) $\mathbf{E}[e(i)^2]$ versus the number of iterations i. The two main ways to obtain the estimate of $\mathbf{E}[e(i)^2]$ include the *ensemble-average* approach and the testing mean-square-error approach.

To obtain the ensemble-averaged learning curve, we need an ensemble of adaptive filters, with each filter operating with the same configuration settings such as updating rule, step-size parameter, and initialization. The input and desired signals are independent for each filter. For each filter, we plot the *sample* learning curve, which is simply the squared value of the estimation error $e(i)^2$ (notice there is no expectation operator here) versus the number of iterations. The sample learning curve so obtained consists of noisy components because of the inherently stochastic nature of the adaptive filter. Then we take the average of these sample learning curves over the ensemble of adaptive filters used in the experiment, thereby smoothing out the effects of noise. The averaged learning curve is called the ensemble-averaged learning curve. This method is applicable for any environment, stationary or nonstationary.

The other approach is by setting aside a testing data set before the training. For each iteration, we have the weight estimate $\mathbf{w}(i)$.We compute the mean square error on the testing data set by using $\mathbf{w}(i)$. Then, we plot the testing MSE versus the number of iterations. This approach only needs one adaptive filter and is computationally cheaper comparing with the ensemble-average approach. However, this method does not apply in situations where the environment is nonstationary.

2.2 KERNEL LEAST-MEAN-SQUARE ALGORITHM

A *linear finite impulse response filter* is assumed in the LMS algorithm. If the mapping between d and $\mathbf{u}$ is highly nonlinear, then poor performance can be expected from LMS. To overcome the limitation of linearity, we are well motivated to formulate a "similar" algorithm that is capable of learning arbitrary nonlinear mappings. For that purpose, the kernel-induced mapping

[equation (1.27)] is employed to transform the input $\mathbf{u}(i)$ into a high-dimensional feature space $\mathbb{F}$ as $\boldsymbol{\varphi}(\mathbf{u}(i))$. As we discussed in Chapter 1, $\boldsymbol{\omega}^T\boldsymbol{\varphi}(\mathbf{u})$ is a much more powerful model than $\mathbf{w}^T\mathbf{u}$ because of the difference in dimensionality (more importantly the richness of representation) of $\mathbf{u}$ and $\boldsymbol{\varphi}(\mathbf{u})$. So, finding $\boldsymbol{\omega}$ through stochastic gradient descent may prove as an effective way of nonlinear filtering as LMS does for linear problems. Denote $\boldsymbol{\varphi}(i) = \boldsymbol{\varphi}(\mathbf{u}(i))$ for simplicity. Using the LMS algorithm on the new example sequence $\{\boldsymbol{\varphi}(i), d(i)\}$ yields

$$\begin{aligned}
&\boldsymbol{\omega}(0) = 0 \\
&e(i) = d(i) - \boldsymbol{\omega}(i-1)^T\boldsymbol{\varphi}(i) \\
&\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta e(i)\boldsymbol{\varphi}(i)
\end{aligned} \tag{2.13}$$

where $\boldsymbol{\omega}(i)$ denotes the estimate (at iteration i) of the weight vector in $\mathbb{F}$. We can see the direct correspondence between equations (2.1) and (2.13).

However, the dimensionality of $\boldsymbol{\varphi}$ is high (infinity in the case of the Gaussian kernel) and $\boldsymbol{\varphi}$ is only implicitly known (it is the kernel's eigenfunctions), so we need an alternative way of carrying out the computation. The repeated application of the weight-update equation (2.13) through iterations yields

$$\begin{aligned}
\boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta e(i)\boldsymbol{\varphi}(i) \\
&= [\boldsymbol{\omega}(i-2) + \eta e(i-1)\boldsymbol{\varphi}(i-1)] + \eta e(i)\boldsymbol{\varphi}(i) \\
&= \boldsymbol{\omega}(i-2) + \eta[e(i-1)\boldsymbol{\varphi}(i-1) + e(i)\boldsymbol{\varphi}(i)] \\
&\cdots \\
&= \boldsymbol{\omega}(0) + \eta\sum_{j=1}^{i} e(j)\boldsymbol{\varphi}(j) \\
&= \eta\sum_{j=1}^{i} e(j)\boldsymbol{\varphi}(j) \quad (\text{assuming } \boldsymbol{\omega}(0) = 0)
\end{aligned} \tag{2.14}$$

that is, after i-step training, the weight estimate is expressed as a linear combination of all the previous and present (transformed) inputs, weighted by the prediction errors (and scaled by η). More importantly, the output of the system to a new input $\mathbf{u}'$ can be solely expressed in terms of inner products between transformed inputs

$$\begin{aligned}
\boldsymbol{\omega}(i)^T\boldsymbol{\varphi}(\mathbf{u}') &= \left[\eta\sum_{j=1}^{i} e(j)\boldsymbol{\varphi}(\mathbf{u}(j))^T\right]\boldsymbol{\varphi}(\mathbf{u}') \\
&= \eta\sum_{j=1}^{i} e(j)\left[\boldsymbol{\varphi}(\mathbf{u}(j))^T\boldsymbol{\varphi}(\mathbf{u}')\right]
\end{aligned} \tag{2.15}$$

Now by the kernel trick [equation (1.28)] we can efficiently compute the filter output in the input space by kernel evaluations

$$\boldsymbol{\omega}(i)^T \boldsymbol{\varphi}(\mathbf{u}') = \eta \sum_{j=1}^{i} e(j)\kappa(\mathbf{u}(j), \mathbf{u}') \tag{2.16}$$

It is important to stop here and compare this equation with the weight update of LMS [equation (2.1)]. The new algorithm is computed without using the weights. Instead, we have the sum of all past errors multiplied by the kernel evaluations on the previously received data, which is equivalent to the weights as can be observed in [equation (2.14)]. Therefore, having direct access to the weights enables the computation of the output with a single inner product, which is a huge time saving, but the two procedures are actually equivalent.

If f_i is denoted as the estimate of the input–output nonlinear mapping at time i, we have the following sequential learning rule for the new algorithm:

$$\begin{aligned} f_{i-1} &= \eta \sum_{j=1}^{i-1} e(j)\kappa(\mathbf{u}(j), \cdot) \\ f_{i-1}(\mathbf{u}(i)) &= \eta \sum_{j=1}^{i-1} e(j)\kappa(\mathbf{u}(j), \mathbf{u}(i)) \\ e(i) &= d(i) - f_{i-1}(\mathbf{u}(i)) \\ f_i &= f_{i-1} + \eta e(i)\kappa(\mathbf{u}(i), \cdot) \end{aligned} \tag{2.17}$$

We call the new algorithm KLMS. It is the LMS in RKHS, and filtering is done by kernel evaluation. KLMS allocates a new kernel unit for the new training data with input $\mathbf{u}(i)$ as the center and $\eta e(i)$ as the coefficient. The coefficients and the centers are stored in memory during training. The algorithm is summarized in algorithm 2 and illustrated in Figure 2.1. $\mathbf{a}(i)$ is the

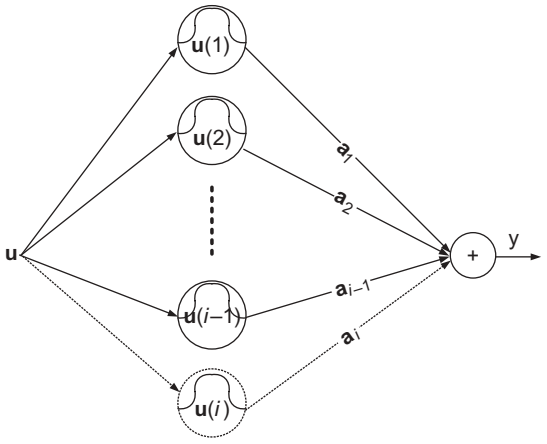


Figure 2.1. Network topology of KLMS at iteration *i*.

coefficient vector at iteration i, $\mathbf{a}_j(i)$ is its jth component, and $\mathcal{C}(i)$ is the corresponding set of centers. At iteration i, given a test input point $\mathbf{u}_*$, the output of the system is

$$f(\mathbf{u}_*) = \eta \sum_{j=1}^{i} e(j)\kappa(\mathbf{u}(j), \mathbf{u}_*) \tag{2.18}$$

The KLMS topology reminds us of a radial-basis function (RBF) network,[1] with three major differences: First, the output weights are essentially the scaled prediction errors at each sample; second, this is a *growing network* where each new unit is placed over each new input; third, κ is not limited to be a radial-basis function and can be any Mercer kernel.

KLMS is a simple algorithm, which requires $O(i)$ operations per filter evaluation and weight update, but we need to pay attention to several aspects that are still unspecified. The first is how to select the kernel κ; the second is how to select the step-size parameter η; and finally, the third is how to cope with the growing memory/computation requirement for online operation.

Algorithm 2. The kernel least-mean-square algorithm

Initialization

choose step-size parameter η and kernel κ
$\mathbf{a}_1(1) = \eta d(1)$, $\mathcal{C}(1) = \{\mathbf{u}(1)\}$, $f_1 = \mathbf{a}_1(1)\kappa(\mathbf{u}(1), \cdot)$

Computation

while $\{\mathbf{u}(i), d(i)\}$ available **do**
  %compute the output
  $f_{i-1}(\mathbf{u}(i)) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa(\mathbf{u}(i), \mathbf{u}(j))$
  %compute the error
  $e(i) = d(i) - f_{i-1}(\mathbf{u}(i))$
  %store the new center
  $\mathcal{C}(i) = \{\mathcal{C}(i-1), \mathbf{u}(i)\}$
  %compute and store the coefficient
  $\mathbf{a}_i(i) = \eta e(i)$
end while

2.3 KERNEL AND PARAMETER SELECTION

The necessity of specifying the kernel and its parameter applies to all kernel methods, and it is reminiscent of nonparametric regression, where the weight function and its smoothing parameter must be chosen. The kernel is a crucial ingredient of any kernel method in the sense that it defines the *similarity* between data points. An exhaustive treatment on this topic is out of the scope

of the book.[2] In the following section, we provide a brief and engineering-oriented discussion.

First and foremost, we need to pick a kernel. In the literature of nonparametric regression, it is known that any bell-shaped weight function (Gaussian function, tricube function, etc.) leads to equivalent asymptotic accuracy. However, weight functions are not necessarily reproducing kernels and vice versa. For example, the polynomial kernel [equation (1.25)] is not bell-shaped and cannot be considered as a weight function. The RKHS approach examines more closely the eigenfunctions of the kernel and its richness for approximation. It is known that the Gaussian kernel (among many others such as the Laplacian) creates a reproducing kernel Hilbert space with universal approximating capability, whereas the polynomial kernel of finite order does not. The approximating capability of the polynomial kernel with order p is limited to any polynomial function with its degree less than or equal to p. Unless it is clear from the problem domain that the target function is a polynomial function or can be well approximated by a polynomial function, the Gaussian kernel is usually a default choice. The Gaussian kernel has the *universal approximating capability*, is numerically stable, and usually gives reasonable results.

The *kernel bandwidth* (also known as kernel size or smoothing parameter) in the Gaussian kernel is an important parameter to be specified. In Chapter 1, we defined the Gaussian kernel as

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp\left(-a\|\mathbf{u} - \mathbf{u}'\|^2\right) \tag{2.19}$$

which is sometimes defined as

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp\left(-\frac{\|\mathbf{u} - \mathbf{u}'\|^2}{2h^2}\right) \tag{2.20}$$

where h is the kernel bandwidth. If the Gaussian kernel is defined as in equation (2.19), the kernel bandwidth is $h = 1/\sqrt{2a}$. And a is simply called the kernel parameter. The available methods to select suitable kernel bandwidth include cross-validation, nearest neighbors, penalizing functions, and plug-in methods [Härdle, 1992]. From the viewpoint of functional analysis, the kernel size helps define the inner product, i.e., the metric of similarity in RKHS. Similarity is the basis of inference. Therefore, the same input data can be mapped to vastly different functionals depending on the kernel bandwidth selected. And different filter outputs will be created if the kernel bandwidth is varied on the same data with the same kernel. If the kernel size is too large, then all the data would look similar in the RKHS (with inner products all close to 1), and the system reduces to linear regression. If the kernel size is too small, then all the data would look distinct (with inner products all close to 0) and the system cannot do inference on unseen samples that fall between the training points. Because the kernel size is a free parameter and we are

interested in an adaptive framework, potentially it may be adapted during operation as any other parameter. The resource allocating network is such an example and other relevant work can be found in the literature of locally adaptive kernel regression estimation [Herrmann, 1997]. In nonparametric regression, the kernel size is usually framed as the compromise between the mean and the variance of the estimator, which is appropriate to help us find experimental procedures to estimate its optimal value from the data. Our experience tells that *cross-validation* on a small subset of data is usually adequate to select an appropriate kernel bandwidth and it is very straightforward. We provide a brief introduction on cross-validation below, and more details can be found in Wahba [1990].

If data are abundant and a validation set is affordable, then the cross-validation (CV) cost function is defined as

$$\mathrm{CV}(h) = N_{\mathrm{CV}}^{-1} \sum_{j=1}^{N_{\mathrm{CV}}} \left[y_j - f_h(\mathbf{x}_j) \right]^2 \tag{2.21}$$

where h is the parameter we need to choose, $\{(\mathbf{x}_j, y_j)\}_{j=1}^{N_{\mathrm{CV}}}$ is the validation set, and f_h is the estimated function by using the training data and h. We are interested in the minimum of this curve across a range of h values. If training data are scarce, then *k-fold cross-validation* can be used. First, the training data are randomly split into k disjoint, equally sized subsets. Then, each subset is picked as a validation set and the training is done on the union of the remaining $k - 1$ subsets. After this process is repeated k times by using different subset as validation set, we get k systems with k cross-validation cost, which is denoted by $\mathrm{CV}_1(h), \mathrm{CV}_2(h), \ldots, \mathrm{CV}_k(h)$. Therefore, the overall cost function of the k-fold cross-validation is

$$k\mathrm{CV}(h) = k^{-1} \sum_{j=1}^{k} \mathrm{CV}_j(h) \tag{2.22}$$

An extreme case of k-fold cross-validation is *leave-one-out cross-validation* (LOOCV), where k equals the number of training data. The cost function of the leave-one-out cross-validation can be simply expressed as

$$\mathrm{LOOCV}(h) = N^{-1} \sum_{j=1}^{N} \left[y_j - f_{h,j}(\mathbf{x}_j) \right]^2 \tag{2.23}$$

where $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ is the training set and $f_{h,j}$ is the estimated function by using the training data excluding only the jth pair $\{(\mathbf{x}_j, y_j)\}$.

Searching for the best value by cross-validation is simple but can be tedious. It would be nice if we have a rough guess to start with. If $\mathbf{x}$ is one-dimensional, then Silverman's rule is often helpful:

$$h_s = 1.06 \min\{\sigma, R/1.34\} N^{-1/5} \tag{2.24}$$

where σ is the standard deviation of $\mathbf{x}$ and R is the interquartile of $\mathbf{x}$. The range $[h_s/10, 10h_s]$ is a good start for cross-validation. If $\mathbf{x}$ is multidimensional, then it gets more complicated. In many nonlinear signal processing applications, $\mathbf{x}$ is constructed by time-embedding a one-dimensional time series. Then, we can roughly estimate that the optimal parameter is somewhere in the interval

$$\left[1.06\min\{\sigma, R/1.34\}N^{-1/5}, 1.06\min\{\sigma, R/1.34\}N^{-1/(5L)}\right]$$

where σ is the standard deviation of the time series, R is the interquartile of the time series, and L is the time-embedding dimension. This is understandable because higher dimensionality requires far more data to cover the data space. We have to emphasize that choosing the parameter requires experience and experiments.

2.4 STEP-SIZE PARAMETER

After choosing the kernel and its free parameter, the next thing is to find a suitable step-size parameter. Because KLMS is the LMS algorithm in RKHS, the role of the step-size parameter remains in principle the same and the results from the adaptive filtering literature can be used. In particular, the step-size parameter is the compromise between convergence time and misadjustment (i.e., increasing the step-size parameter decreases convergence time but increases misadjustment). Moreover, the step-size parameter is upper bounded by the reciprocal of the largest eigenvalue of the transformed data autocorrelation matrix. Denoting the transformed data matrix $\Phi = [\boldsymbol{\varphi}(1), \boldsymbol{\varphi}(2), \ldots, \boldsymbol{\varphi}(N)]$, $\mathbf{R}_{\boldsymbol{\varphi}}$ its autocorrelation matrix, and $\mathbf{G}_{\boldsymbol{\varphi}}$ its Gram matrix, we have

$$\begin{aligned}\mathbf{R}_{\boldsymbol{\varphi}} &= \frac{1}{N}\boldsymbol{\Phi}\boldsymbol{\Phi}^T \\ \mathbf{G}_{\boldsymbol{\varphi}} &= \boldsymbol{\Phi}^T\boldsymbol{\Phi}\end{aligned} \tag{2.25}$$

$\mathbf{G}_{\boldsymbol{\varphi}}$ is an $N \times N$ matrix with $\kappa(\mathbf{u}(i), \mathbf{u}(j))$ as its (i, j)-th component.

The step-size parameter is required to satisfy the following condition for the algorithm to stay stable [Haykin, 2002]:

$$\eta < \frac{1}{\varsigma_{\max}} \tag{2.26}$$

where $\varsigma_{\max}$ is the largest eigenvalue of $\mathbf{R}_{\boldsymbol{\varphi}}$. The dimensionality of $\mathbf{R}_{\boldsymbol{\varphi}}$ could be high, and it is usually unfeasible to compute it directly. Fortunately, its eigenvalues can be computed from $\mathbf{G}_{\boldsymbol{\varphi}}$ [Golub and Loan, 1996]. More specifically, if $\mathbf{R}_{\boldsymbol{\varphi}}$ has r nonzero eigenvalues $\{\varsigma_j\}_{j=1}^{r}$, then $\mathbf{G}_{\boldsymbol{\varphi}}$ also has r nonzero eigenvalues,

which are $\{N\varsigma_j\}_{j=1}^{r}$. Because $\mathbf{R}_\varphi$ and $\mathbf{G}_\varphi$ are both positive semidefinite, all the nonzero eigenvalues are positive. It is also known that the trace of a matrix equals the summation of all its eigenvalues. Using these facts, we have

$$\varsigma_{\max} < \mathrm{tr}[\mathbf{R}_\varphi] = \mathrm{tr}[\mathbf{G}_\varphi]/N$$

Therefore, a conservative upper bound for the step-size parameter is

$$\eta < \frac{N}{\mathrm{tr}[\mathbf{G}_\varphi]} = \frac{N}{\sum_{j=1}^{N} \kappa(\mathbf{u}(j), \mathbf{u}(j))} \tag{2.27}$$

For shift-invariant kernels, i.e., $\kappa(\mathbf{u}(j), \mathbf{u}(j)) = g_0$, the upper bound becomes $1/g_0$, which is data independent. We find this upper bound is handy in practice and use it as a default value.

Other properties of the LMS algorithm can also be easily used for KLMS. For example, the misadjustment of KLMS can be estimated as

$$\mathcal{M} = \frac{\eta}{2}\mathrm{tr}[\mathbf{R}_\varphi] = \frac{\eta}{2N}\mathrm{tr}[\mathbf{G}_\varphi] \tag{2.28}$$

In the case of shift-invariant kernels, the misadjustment of KLMS equals $\eta g_0/2$, which is also data-independent and is simply proportional to the step-size parameter.

2.5 NOVELTY CRITERION

In a stationary environment, the learning system will eventually converge after processing sufficient examples and will stop training afterward. As we observe in the formulation of KLMS, the size of the network increases linearly with the number of training data, which poses a challenge for applying KLMS in nonstationary signal processing. A fundamental question is whether it is necessary to memorize all the past inputs. By removing redundant data, it is possible to keep a minimal set of centers that covers the area where inputs will likely appear [imagine that a kernel is a sphere in the input space ($\mathbb{R}^L$) with the kernel bandwidth as the radius]. On the other hand, a sparse model (a network with as few kernels as possible) is desirable because it reduces the complexity in terms of computation and memory, and it usually gives better generalization ability (Occam's Razor). There are many approaches to sparsification of kernel-based solutions, but most of them are off-line methods. We focus the discussion here on *online sparsification* or *sequential sparsification*. A simple way to check whether the newly arrived datum is informative enough is the *novelty criterion* (NC) proposed by Platt [1991]. Richard et al. [2009] also studied a similar method called *coherence criterion* with many mathematical

properties. Engel et al. [2004] introduced another way to tackle this problem with the idea of *approximate linear dependency* (ALD) test, which is close to the work of Csato and Opper [2002]. This has also been explored specifically for KLMS in Pokharel et al. [2009]. Sequential sparsification is also being studied in computational learning theory, such as the kernel perceptron with a fixed budget [Dekel et al., 2006]. We will propose yet another criterion to address this issue in Chapter 6 and unify NC and ALD in a rigorous information theoretic framework. In this chapter, we focus on Platt's novelty criterion.

Online sparsification is usually obtained by *construction* in a sense that it starts from an empty set and gradually adds samples into a center set called the *dictionary* according to some criterion. Suppose the present dictionary is $\mathcal{C}(i) = \{\mathbf{c}_j\}_{j=1}^{m_i}$, where $\mathbf{c}_j$ is the jth center and m_i is the cardinality. When a new data pair $\{\mathbf{u}(i+1), d(i+1)\}$ is presented, a decision is immediately made of whether $\mathbf{u}(i+1)$ should be added into the dictionary as a new center. In novelty criterion, it first calculates the distance of $\mathbf{u}(i+1)$ to the present dictionary $\text{dis}_1 = \min_{\mathbf{c}_j \in \mathcal{C}(i)} \|\mathbf{u}(i+1) - \mathbf{c}_j\|$. If it is smaller than some preset threshold, say δ_1, $\mathbf{u}(i+1)$ will not be added into the dictionary. Otherwise, the algorithm computes the prediction error $e(i+1)$. Only if the prediction error is larger than another preset threshold, say δ_2, will $\mathbf{u}(i+1)$ be accepted as a new center. Here are some heuristics on how to set the parameters for NC. Initially, the kernel filter is designed without the novelty criterion such that we can focus on step-size parameter and kernel size selection. After picking the kernel size and having an estimate of steady-state MSE, the second step becomes straightforward. A reasonable δ_1 is around one tenth of the kernel bandwidth $\sqrt{1/2a}$. Increasing δ_1 will decrease the network size, but the performance may degrade. A reasonable default value for δ_2 is the square root of the steady-state MSE. Increasing δ_2 will decrease the network size, but the performance may degrade. Cross-validation also can be used to select appropriate thresholds.

If the input domain $\mathbb{U}$ is a compact set, with the aid of the novelty criterion, the cardinality of the dictionary is always finite and upper bounded. This statement is not hard to prove using the finite covering theorem of the compact set and the fact that elements in the dictionary are δ-separable. The following section is a brief outline of the proof.

Suppose spheres with diameter δ are used to cover $\mathbb{U}$ and the optimal covering number is N_c. Then, because any two centers in the dictionary cannot be in the same sphere, the total number of the centers will be no greater than N_c regardless of the distribution and temporal structure of $\mathbf{u}$. Of course, this is a worst-case upper bound. In the case of finite training data, the network size will be finite anyway. This is true in applications like channel equalization, where the training sequence is part of each transmission frame. In a stationary environment, the network converges quickly and the threshold on prediction errors plays its part to constrain the network size. We will validate this claim in the simulation section. In a nonstationary environment, there are two

scenarios. In the first scenario, the input domain does not change and only the input–output mapping changes. After the network grows to a point that the input domain is sufficiently covered, simple LMS can be used to just modify the coefficients to track the nonstationarity. In the second scenario, input domain changes as well. In this case, pruning methods should be used to constrain the network size.[3] An alternative approach is to solve the problem in the primal space directly by using the low-rank approximation methods.[4] It should be pointed out that the scalability issue is at the core of the kernel methods, and all the kernel methods need to deal with it in one way or the other. Indeed, the sequential nature of KLMS enables active learning on huge data sets.

2.6 SELF-REGULARIZATION PROPERTY OF KLMS

The KLMS algorithm is derived in a high-dimensional feature space, using a stochastic gradient to solve a least-squares problem. If we study any other kernel machine algorithms, we are alerted for the central role of regularization to obtain solutions that generalize appropriately. Therefore, it is not surprising that all the attempts to derive kernel adaptive filters mentioned in Chapter 1 used a regularized cost function. The surprising fact is that we were able to prove mathematically that KLMS does not need explicit regularization because it is well posed in the sense of Hadamard [Liu et al., 2008]. These results are summarized below.

2.6.1 Solution Norm Bound

From the viewpoints of the regularization and optimization theories [Hoerl and Kennard, 1970], the concepts of regularization, stability, and solution norm constraint are tightly related. The significance of an upper bound for the solution norm is also studied by Poggio and Smale [2003].

Assume the training data $\{\mathbf{u}(i), d(i)\}_{i=1}^{N}$ satisfy a multiple linear regression model in the RKHS:

$$d(i) = \boldsymbol{\varphi}(i)^T \boldsymbol{\omega}^o + v(i) \tag{2.29}$$

where $\boldsymbol{\omega}^o$ is the underlying model and $v(i)$ is the modeling uncertainty. Then by the H^∞ robustness theorem [Haykin, 2002]: For any unknown vector $\boldsymbol{\omega}^o$ and finite energy noise sequence $v(i)$ without further statistical assumptions, the following inequality holds:

$$\frac{\sum_{j=1}^{i} |\hat{s}(j) - s(j)|^2}{\eta^{-1} \left\|\boldsymbol{\omega}^{\mathrm{o}}\right\|^2 + \sum_{j=1}^{i-1} |v(j)|^2} < 1 \tag{2.30}$$

if and only if the matrices $\{\eta^{-1}I - \boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T\}$ are positive definite for all $i \le N$. In the inequality, $s(j) = (\boldsymbol{\omega}^o)^T\boldsymbol{\varphi}(j)$ and $\hat{s}(j) = \boldsymbol{\omega}(j-1)^T\boldsymbol{\varphi}(j)$, where $\boldsymbol{\omega}(j-1)$ is calculated by the KLMS recursion [equation (2.13)]. This result is used to prove the following theorem.

Theorem 2.1. *Under the H^∞ stability condition, the prediction error satisfies the following inequality:*

$$\|\mathbf{e}\|^2 < \eta^{-1}\|\boldsymbol{\omega}^o\|^2 + 2\|\mathbf{v}\|^2 \tag{2.31}$$

where $\mathbf{e} = [e(1), \ldots, e(N)]^T$ *and* $\mathbf{v} = [v(1), \ldots, v(N)]^T$.

Proof. First, we have

$$e(i) - v(i) = s(i) - \hat{s}(i)$$

Substituting it into equation (2.30), we have

$$\frac{\sum_{j=1}^{i}|e(j)-v(j)|^2}{\eta^{-1}\|\boldsymbol{\omega}^o\|^2 + \sum_{j=1}^{i-1}|v(j)|^2} < 1$$

or equivalently,

$$\sum_{j=1}^{i}|e(j)-v(j)|^2 < \eta^{-1}\|\boldsymbol{\omega}^o\|^2 + \sum_{j=1}^{i-1}|v(j)|^2$$

By the triangle inequality

$$\sum_{j=1}^{i}|e(j)|^2 \le \sum_{j=1}^{i}|e(j)-v(j)|^2 + \sum_{j=1}^{i}|v(j)|^2 < \eta^{-1}\|\boldsymbol{\omega}^o\|^2 + \sum_{j=1}^{i-1}|v(j)|^2 + \sum_{j=1}^{i}|v(j) \tag{2.32}$$

which is valid for all $i \le N$. In terms of vector norm,

$$\|\mathbf{e}\|^2 < \eta^{-1}\|\boldsymbol{\omega}^o\|^2 + 2\|\mathbf{v}\|^2 \tag{2.33}$$

□

Theorem 2.2. *Under the H^∞ stability condition, $\boldsymbol{\omega}(N)$ is upper bounded:*

$$\|\boldsymbol{\omega}(N)\| < \sqrt{N\varsigma_1\eta\left(\|\boldsymbol{\omega}^o\|^2 + 2\eta\|\mathbf{v}\|^2\right)} \tag{2.34}$$

where ς_1 is the largest eigenvalue of $\mathbf{R}_\varphi$.

Proof.

$$\begin{aligned}\|\boldsymbol{\omega}(N)\|^2 &= \left\|\eta\sum_{i=1}^{N} e(i)\boldsymbol{\varphi}(i)\right\|^2 \\ &= \eta^2\mathbf{e}^T\mathbf{G}_{\boldsymbol{\varphi}}\mathbf{e} \\ &= \eta^2 N\mathbf{e}^T\mathbf{Q}\mathrm{diag}\{\varsigma_1, \varsigma_2, \ldots, \varsigma_N\}\mathbf{Q}^T\mathbf{e} \\ &\le \eta^2 N\mathbf{e}^T\mathbf{Q}\mathrm{diag}\{\varsigma_1, \varsigma_1, \ldots, \varsigma_1\}\mathbf{Q}^T\mathbf{e} \\ &= \eta^2 N\varsigma_1\|\mathbf{Q}^T\mathbf{e}\|^2 \\ &= \eta^2 N\varsigma_1\|\mathbf{e}\|^2\end{aligned}$$

where

$$\mathbf{G}_{\boldsymbol{\varphi}} = \mathbf{Q}\mathrm{diag}\{N\varsigma_1, N\varsigma_2, \ldots, N\varsigma_N\}\mathbf{Q}^T$$

is the standard eigenvalue decomposition. $\mathbf{Q}$ is an orthogonal matrix. Then, by Theorem 2.1, we have the result directly. □

This result effectively shows that the norm of the KLMS solution is constrained. It also directly implies the compactness of the hypothesis space and thus ensures algorithmic stability.

2.6.2 Singular Value Analysis

Although the result in Theorem 2.2 is conclusive, several useful insights have been neglected. Meanwhile, a singular value analysis can clearly show that the self-regularization property of KLMS is caused by its different convergence speeds along different eigendirections.

Let the singular value decomposition (SVD) of Φ be

$$\boldsymbol{\Phi} = \mathbf{P}\begin{bmatrix}\mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{Q}^T \tag{2.35}$$

where $\mathbf{P}$, and $\mathbf{Q}$ are orthogonal matrices and $\mathbf{S} = \mathrm{diag}(s_1, \ldots, s_r)$ with s_i the singular values and r the rank of Φ. It is assumed that $s_1 \ge \ldots \ge s_r > 0$ without loss of generality. Then, we have

$$\mathbf{R}_{\boldsymbol{\varphi}} = \mathbf{P}\begin{bmatrix}\mathbf{S}^2/N & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{P}^T \tag{2.36}$$

$$\mathbf{G}_{\boldsymbol{\varphi}} = \mathbf{Q}\begin{bmatrix}\mathbf{S}^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{Q}^T \tag{2.37}$$

It is clear that $\varsigma_j = s_j^2/N$.

The well-known pseudo-inverse solution to estimate ω^o in equation (2.29) obtained by minimizing

$$J(\boldsymbol{\omega}) = \left\| \mathbf{d} - \boldsymbol{\Phi}^T \boldsymbol{\omega} \right\|^2 \tag{2.38}$$

is

$$\boldsymbol{\omega}_{PI} = \mathbf{P}\text{diag}\left(s_1^{-1}, \ldots, s_r^{-1}, 0, \ldots, 0\right)\mathbf{Q}^T\mathbf{d} \tag{2.39}$$

The least-squares solution (even with the pseudo-inverse, just think of a very small s_r) can be ill-posed because of the nature of the problem, small data size, or severe noise. The Tikhonov regularization [Tikhonov and Arsenin, 1977] is used widely to address this issue. A regularization term is introduced in the least-squares cost function, which penalizes the solution norm

$$J(\boldsymbol{\omega}) = \left\| \mathbf{d} - \boldsymbol{\Phi}^T \boldsymbol{\omega} \right\|^2 + \lambda \left\| \boldsymbol{\omega} \right\|^2 \tag{2.40}$$

Solving this minimization problem yields the Tikhonov regularization solution

$$\boldsymbol{\omega}_{TR} = \mathbf{P}\text{diag}\left(\frac{s_1}{s_1^2 + \lambda}, \ldots, \frac{s_r}{s_r^2 + \lambda}, 0, \ldots, 0 \right)\mathbf{Q}^T\mathbf{d} \tag{2.41}$$

Comparing equation (2.41) with equation (2.39), we observe that the Tikhonov regularization modifies the diagonal terms through the following regularization function (reg-function):

$$H_{TR}(x) = \frac{x^2}{x^2 + \lambda} \tag{2.42}$$

If s_r is small, then the pseudo-inverse solution becomes problematic as the solution approaches infinity. However, for the Tikhonov regularization, $H_{TR}(s_r)s_r^{-1} \to 0$ if s_r is small and $H_{TR}(s_r)s_r^{-1} \to s_r^{-1}$ if s_r is large. In this sense, the Tikhonov regularization smoothly filters out the minor components that correspond to small singular values (relative to λ). Attenuating the minor components is important to get a smaller norm solution or, in other words, a more stable solution. With this understanding, the so-called truncated pseudo-inverse regularization [Golub and Loan, 1996] is nothing but using the following hard cut-off reg-function:

$$H_{PCA}(x) = \begin{cases} 1 & \text{if } x > t \\ 0 & \text{if } x \le t \end{cases} \tag{2.43}$$

where t is the cut-off threshold. If $s_k > t \ge s_{k+1}$ (usually $k << r$), the solution becomes

$$\boldsymbol{\omega}_{PCA} = \mathbf{P}\text{diag}\left(s_1^{-1}, \ldots, s_k^{-1}, 0, \ldots, 0\right)\mathbf{Q}^T\mathbf{d} \tag{2.44}$$

This method is equivalent to applying the principal components analysis (PCA) technique to the data and using the first k principal components to represent the original data. Under a reasonable signal-to-noise ratio, the small singular value components are purely associated with the noise. Discarding these spurious features can effectively prevent overlearning.

With the above discussion, we are ready to show why KLMS possesses a self-regularization property. First, define the natural modes of the weight error in terms of the eigenvectors of $\mathbf{R}_{\varphi}$:

$$\boldsymbol{\omega}(n) - \boldsymbol{\omega}^{\mathrm{o}} = \sum_{j=1}^{M} \varepsilon_j(n)\mathbf{P}_j$$

where $\mathbf{P}_j$ is the jth column of $\mathbf{P}$, M is the dimensionality of $\mathbf{R}_{\varphi}$, $\varepsilon_j(n)$ denotes the distance between $\boldsymbol{\omega}(n)$ and $\boldsymbol{\omega}^{o}$ in the jth eigenvector direction. It has been shown that [Haykin, 2002]

$$\begin{aligned} \mathbf{E}[\varepsilon_j(n)] &= (1-\eta\varsigma_j)^n \varepsilon_j(0) \\ \mathbf{E}\left[|\varepsilon_j(n)|^2\right] &= \frac{\eta J_{\min}}{2-\eta\varsigma_j} + (1-\eta\varsigma_j)^{2n}\left(|\varepsilon_j(0)|^2 - \frac{\eta J_{\min}}{2-\eta\varsigma_j}\right) \end{aligned} \tag{2.45}$$

where $J_{\min}$ is the irreducible error power. Therefore,

$$\mathbf{E}[\boldsymbol{\omega}(n)] = \boldsymbol{\omega}^{o} + \sum_{j=1}^{M}(1-\eta\varsigma_j)^n \varepsilon_j(0)\mathbf{P}_j \tag{2.46}$$

Furthermore, with $\boldsymbol{\omega}^{o} = \sum_{j=1}^{M} \boldsymbol{\omega}_j^{o}\mathbf{P}_j$, $\boldsymbol{\omega}(0) = 0$, and $\varepsilon_j(0) = -\boldsymbol{\omega}_j^{o}$, we have

$$\begin{aligned} \mathbf{E}[\boldsymbol{\omega}(n)] &= \sum_{j=1}^{M} \boldsymbol{\omega}_j^{o}\mathbf{P}_j - \sum_{j=1}^{M}(1-\eta\varsigma_j)^n \boldsymbol{\omega}_j^{o}\mathbf{P}_j \\ &= \sum_{j=1}^{M}\left[1-(1-\eta\varsigma_j)^n\right]\boldsymbol{\omega}_j^{o}\mathbf{P}_j \end{aligned} \tag{2.47}$$

It is clear that the norm of the expected weight is upper bounded by

$$\begin{aligned} \|\mathbf{E}[\boldsymbol{\omega}(n)]\|^2 &= \sum_{j=1}^{M}\left[1-(1-\eta\varsigma_j)^n\right]^2 \left(\boldsymbol{\omega}_j^{o}\right)^2 \\ &= \sum_{j=1}^{M}\left[1-(1-\eta\varsigma_j)^n\right]^2 \left(\boldsymbol{\omega}_j^{o}\right)^2 \\ &\le \sum_{j=1}^{M}\left(\boldsymbol{\omega}_j^{o}\right)^2 = \left\|\boldsymbol{\omega}^{o}\right\|^2 \end{aligned} \tag{2.48}$$

assuming $\eta \leq 1/\varsigma_j$. In the worst case, by replacing the optimal weight with the pseudo-inverse solution, we have

$$\begin{aligned}\mathbf{E}[\boldsymbol{\omega}(n)] &= \mathbf{P}\mathrm{diag}\left(\left[1-(1-\eta\varsigma_1)^n\right]s_1^{-1},\ldots,\left[1-(1-\eta\varsigma_r)^n\right]s_r^{-1},0,\ldots,0\right)\mathbf{Q}^T\mathbf{d} \\ &= \mathbf{P}\mathrm{diag}\left(\left[1-\left(1-\eta s_1^2/N\right)^n\right]s_1^{-1},\ldots,\left[1-\left(1-\eta s_r^2/N\right)^n\right]s_r^{-1},0,\ldots,0\right)\mathbf{Q}^T\mathbf{d}\end{aligned} \tag{2.49}$$

which means the reg-function for KLMS (in the mean sense) stopped at iteration N is

$$H_{\mathrm{KLMS}}(x) = 1-\left(1-\eta x^2/N\right)^N \tag{2.50}$$

And the following theorem shows that KLMS takes care of the small singular values.

Theorem 2.3. $\lim_{x\to 0} H_{\mathrm{KLMS}}(x)x^{-1} = 0$

Proof.

$$\begin{aligned}H_{\mathrm{KLMS}}(x)x^{-1} &= \frac{1}{x}\left[1-\left(1-\eta x^2/N\right)\right]\left[1+\left(1-\eta x^2/N\right)+\ldots+\left(1-\eta x^2/N\right)^{N-1}\right] \\ &= \eta x/N\left[1+\left(1-\eta x^2/N\right)+\ldots+\left(1-\eta x^2/N\right)^{N-1}\right]\end{aligned}$$

Therefore, it is a polynomial in x and the conclusion follows directly. □

A comparison of three regularization methods is illustrated in Figure 2.2. In the reg-function of Tikhonov regularization, the regularization parameter is chosen as 1. For the reg-function of PCA, $t = 0.5$. For the reg-function of KLMS, $\eta = 0.1$ and $N = 500$. Furthermore, in Figures 2.3 and 2.4, we show the effect of the step-size parameter and data size on the regularization function of KLMS. The figures show clearly that the step-size parameter affects the regularization significantly, whereas the training data size does not as long as N is sufficiently large. This fact is not surprising if we recall the basic mathematical formula

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e$$

The conclusion from this discussion is that the step-size parameter in KLMS plays a similar role as the regularization parameter in explicitly regularized cost functions. Therefore, there is no need for explicit regularization, which simplifies the algorithm implementation tremendously.

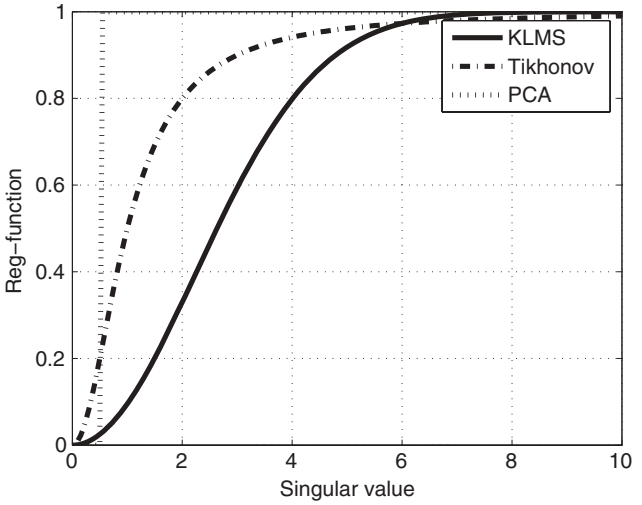


Figure 2.2. Comparison of three regularization approaches: KLMS, Tikhonov regularization, and PCA.

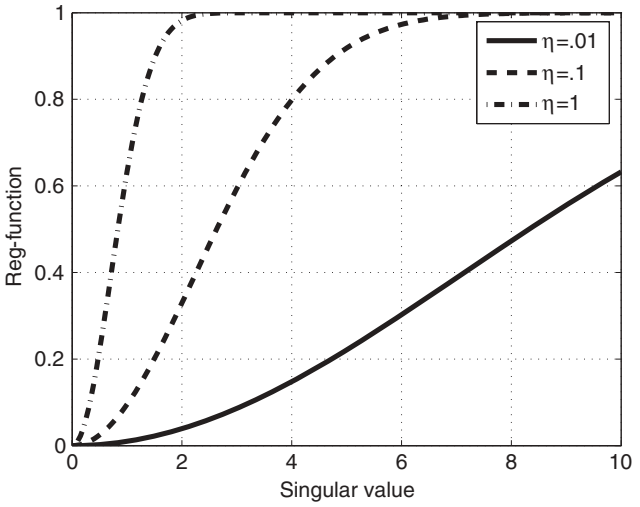


Figure 2.3. Effect of step-size parameter on the reg-function of KLMS (N = 500).

The step-size parameter in KLMS is not only a compromise between adaptation speed and misadjustment, but also it controls the generalization ability of the algorithm. Increasing the step-size parameter leads to a danger of overfitting, whereas a smaller step size helps generalization.

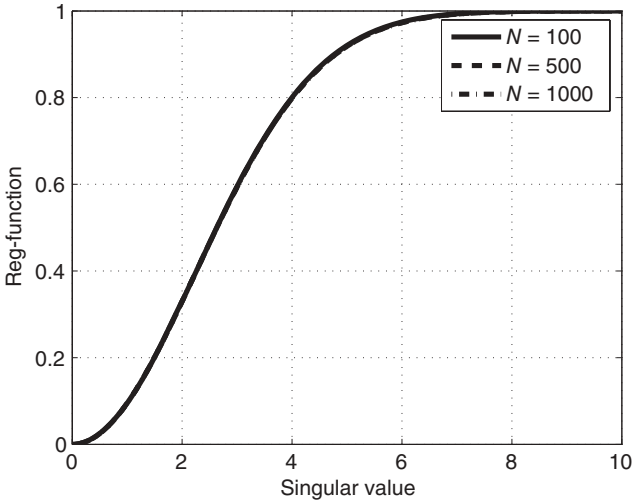


Figure 2.4. Effect of training data size on the reg-function of KLMS ($\eta = 0.1$). Three lines overlap, which means the data size does not affect the shape of the reg-function of KLMS.

2.6.3 A Unit Lower Triangular Linear System

Another interesting observation can be made about KLMS is that it can be formulated as the solution of a unit lower triangular linear system.

Theorem 2.4. *The KLMS prediction errors $e(1)$, $e(2)$, …, $e(i)$ are linearly related to the desired samples $d(1)$, $d(2)$, …, $d(i)$ through a unit lower triangular matrix.*

Proof. By equation (2.16),

$$e(j) = d(j) - \eta \sum_{k=1}^{j-1} e(k)\kappa(\mathbf{u}(k), \mathbf{u}(j))$$

so

$$d(j) = e(j) + \eta \sum_{k=1}^{j-1} e(k)\kappa(\mathbf{u}(k), \mathbf{u}(j))$$

for $j = 1, \ldots, i$. Writing them into matrix form yields

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \eta\kappa_{1,2} & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \eta\kappa_{1,i} & \eta\kappa_{2,i} & \eta\kappa_{3,i} & \cdots & 1 \end{pmatrix}_{i\times i} \begin{pmatrix} e(1) \\ e(2) \\ \cdots \\ e(i) \end{pmatrix}_{i\times 1} = \begin{pmatrix} d(1) \\ d(2) \\ \cdots \\ d(i) \end{pmatrix}_{i\times 1} \tag{2.51}$$

where $\kappa_{i,j} = \kappa(\mathbf{u}(i), \mathbf{u}(j))$ for simplicity. This completes the proof. □

This result is interesting. It tells us that instead of solving a large-scale dense linear system, we can find an "approximate" solution by solving a unit lower triangular linear system. Numerically, we know that inverting a unit lower triangular matrix is stable. When the step-size parameter is small, the matrix is close to the identity matrix and its stability is guaranteed.

2.7 LEAKY KERNEL LEAST-MEAN-SQUARE ALGORITHM

A similar algorithm called NORMA was derived in Kivinen et al. [2004] but from a vastly different viewpoint. Kivinen et al. differentiated directly the following regularized functional to get the stochastic gradient in the function space:

$$\min_f J(f) = \sum_{i=1}^{n} |d(i) - f(\mathbf{u}(i))|^2 + \lambda \|f\|^2$$

with λ as the regularization parameter.

Although the derivation involves advanced mathematics, the results are actually equivalent to the following update rule:

$$f_i = (1 - \eta\lambda) f_{i-1} + \eta e(i) \kappa(\mathbf{u}(i)) \tag{2.52}$$

Comparing equation (2.52) with KLMS equation (2.17), it has a scaling factor $(1 - \eta\lambda)$ on the previous estimate which imposes a forgetting mechanism so that the training data in the far past are scaled down exponentially. Therefore, by neglecting the units with small coefficients, the number of actual active units is finite.

The regularization introduces a bias in the solution as is well known in leaky LMS [Sayed, 2003]. Pokharel et al. [2007] reported that even a small regularization parameter degrades its performance compared with KLMS.

2.8 NORMALIZED KERNEL LEAST-MEAN-SQUARE ALGORITHM

The *normalized least-mean-square algorithm* (NLMS) usually exhibits better performance than LMS in many practical applications. The weight update equation for NLMS is [Haykin, 2002]

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \frac{\eta}{\varepsilon + \|\mathbf{u}(i)\|^2} e(i) \mathbf{u}(i)$$

where $\varepsilon + \|\mathbf{u}(i)\|^2$ is the normalizing term and ε is a small positive number introduced to prevent division-by-zero exception. The normalized LMS algorithm is summarized in Algorithm 3.

Algorithm 3. The normalized least-mean-square algorithm

Initialization

$\mathbf{w}(0) = 0$, choose η, ε

Computation

while $\{\mathbf{u}(i), d(i)\}$ available **do**

$e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i)$

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \frac{\eta}{\varepsilon + \|\mathbf{u}(i)\|^2} e(i)\mathbf{u}(i)$$

end while

It is straightforward to derive the normalized kernel least-mean-square algorithm based on the above discussion. The weight update equation for normalized KLMS is

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \frac{\eta}{\varepsilon + \|\boldsymbol{\varphi}(i)\|^2} e(i)\boldsymbol{\varphi}(i)$$

And by using the definition of the norm in the feature space, we have

$$\|\boldsymbol{\varphi}(i)\|^2 = \langle \boldsymbol{\varphi}(i), \boldsymbol{\varphi}(i) \rangle = \kappa(\mathbf{u}(i), \mathbf{u}(i))$$

If the kernel is shift invariant, i.e., $\kappa(\mathbf{u}(j), \mathbf{u}(j)) = g_0$, then KLMS is automatically normalized.

2.9 KERNEL ADALINE

Kernel ADALINE [Frieb and Harrison, 1999] is a gradient descent method solving an *unregularized* least-squares cost in RKHS. Suppose the number of training data is N and we are solving the following unregularized least-squares cost:

$$\min_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \|\mathbf{d} - \boldsymbol{\Phi}^T \boldsymbol{\omega}\|^2 \tag{2.53}$$

where

$$\boldsymbol{\Phi} = [\boldsymbol{\varphi}(1), \ldots, \boldsymbol{\varphi}(N)]$$
$$\mathbf{d} = [d(1), d(2), \ldots, d(N)]^T$$

The gradient of the cost function [equation (2.53)] is

$$\nabla J(\boldsymbol{\omega}) = -2\boldsymbol{\Phi}\left(\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}\right) \tag{2.54}$$

Therefore, the gradient descent method is

$$\begin{aligned}\boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}\left(\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)\right)/N \\ &= \boldsymbol{\omega}(i-1) + \frac{\eta}{N}\sum_{j=1}^{N}\left[\boldsymbol{\varphi}(j)\left(d(j) - \boldsymbol{\varphi}(j)^T\boldsymbol{\omega}(i-1)\right)\right]\end{aligned} \tag{2.55}$$

where $\boldsymbol{\omega}(i)$ denotes the estimate of the weight at iteration i. η is the step-size parameter. Compared with equation (2.13), it is clear that KLMS is a stochastic gradient descent method, whereas kernel ADALINE is a batch-mode gradient descent method.

With initial value $\boldsymbol{\omega}(0) = 0$, the weight estimate by equation (2.55) is a linear combination of the transformed data at any iteration, i.e.,

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}\mathbf{a}(i) = \sum_{j=1}^{N}\mathbf{a}_j(i)\varphi(j), \quad \forall i \tag{2.56}$$

Notice that this result cannot be derived from the representer theorem because we do not have the explicit norm constraint in equation (2.53). Instead, we can use mathematical induction to prove the claim. Because $\boldsymbol{\omega}(0) = 0$, the claim is true for $i = 0$. Suppose equation (2.56) is true for $i - 1$. Therefore

$$\begin{aligned}\mathbf{e}(i) &= \mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1) \\ &= \mathbf{d} - \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)\mathbf{a}(i-1) \\ &= \mathbf{d} - \mathbf{G}\mathbf{a}(i-1)\end{aligned}$$

Then, by equation (2.55), we have

$$\begin{aligned}\boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}\left(\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)\right)/N \\ &= \boldsymbol{\Phi}\mathbf{a}(i-1) + \eta\boldsymbol{\Phi}\mathbf{e}(i)/N \\ &= \boldsymbol{\Phi}\left(\mathbf{a}(i-1) + \eta\mathbf{e}(i)/N\right)\end{aligned}$$

i.e.,

$$\mathbf{a}(i) = \mathbf{a}(i-1) + \eta\mathbf{e}(i)/N \tag{2.57}$$

This result is crucial in kernel methods, because $\boldsymbol{\omega}$ is in a high-dimensional space and we usually do not have access to it. By writing $\boldsymbol{\omega}$ as a linear combination of the training data, we actually solve a problem with dimensionality N. Furthermore, we can show that the gradient descent iteration of kernel ADALINE provides an inherent regularization similar to KLMS.

First, rewrite equation (2.55) as

$$\begin{aligned}\boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}\left(\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)\right)/N \\ &= \left(\mathbf{I} - \eta\boldsymbol{\Phi}\boldsymbol{\Phi}^T/N\right)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}d/N \\ &= \left(\mathbf{I} - \eta\mathbf{P}\begin{bmatrix}\mathbf{S}^2/N & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{P}^T\right)\boldsymbol{\omega}(i-1) + \eta\mathbf{P}\begin{bmatrix}\mathbf{S}/N & 0 \\ 0 & 0\end{bmatrix}\mathbf{Q}^T\mathbf{d} \\ &= \mathbf{P}\left[\left(\mathbf{I} - \eta\begin{bmatrix}\mathbf{S}^2/N & 0 \\ 0 & 0\end{bmatrix}\right)\left(\mathbf{P}^T\boldsymbol{\omega}(i-1)\right) + \eta\begin{bmatrix}\mathbf{S}/N & 0 \\ 0 & 0\end{bmatrix}\mathbf{Q}^T\mathbf{d}\right] \end{aligned} \tag{2.58}$$

Here, we use the result of equation (2.35). Denote $\mathbf{b}(i) = \mathbf{P}^T\boldsymbol{\omega}(i)$, which amounts to decomposing the weight vector along the column vectors of matrix P as

$$\boldsymbol{\omega}(i) = \sum_{j=1}^{M}\mathbf{b}_j(i)\mathbf{P}_j = \mathbf{P}\mathbf{b}(i)$$

where M is assumed the dimensionality of RKHS. Therefore, by equation (2.58), we have

$$\mathbf{b}(i) = \left(\mathbf{I} - \eta\begin{bmatrix}\mathbf{S}^2/N & 0 \\ 0 & 0\end{bmatrix}\right)\left(\mathbf{b}(i-1)\right) + \eta\begin{bmatrix}\mathbf{S}/N & 0 \\ 0 & 0\end{bmatrix}\mathbf{Q}^T\mathbf{d} \tag{2.59}$$

or equivalently for each component

$$\mathbf{b}_j(i) = \left(1 - \eta s_j^2/N\right)\mathbf{b}_j(i-1) + \eta s_j\mathbf{Q}_j^T\mathbf{d}/N \tag{2.60}$$

for $1 \le j \le M$.

Observe that if $s_j = 0$, then

$$\mathbf{b}_j(i) = \mathbf{b}_j(i-1) = \ldots = \mathbf{b}_j(0)$$

If $s_j \neq 0$, we repeatedly use equation (2.60) for $i = 1, 2, \ldots$ and obtain

$$\begin{aligned}\mathbf{b}_j(i) &= \left(1 - \eta s_j^2/N\right)^i\mathbf{b}_j(0) + \left(\eta s_j\mathbf{Q}_j^T\mathbf{d}/N\right)\left(\sum_{m=0}^{i-1}\left(1 - \eta s_j^2/N\right)^m\right) \\ &= \left(1 - \eta s_j^2/N\right)^i\mathbf{b}_j(0) + \left[1 - \left(1 - \eta s_j^2/N\right)^i\right]\left(\mathbf{Q}_j^T\mathbf{d}\right)/s_j \end{aligned} \tag{2.61}$$

Notice that s_j^2/N is the eigenvalue of the correlation matrix, which is asymptotically independent of N. The interesting observation is if proper early stopping is used in the training, then the solution norm of the kernel ADALINE is upper bounded. For example, we start from $\boldsymbol{\omega}(0) = 0$ and the training stops after n steps. Therefore,

$$\mathbf{b}_j(n) = \left[1 - \left(1 - \eta s_j^2/N\right)^n\right]\left(\mathbf{Q}_j^T\mathbf{d}\right)/s_j \tag{2.62}$$

This equation shows that along different eigendirections, the algorithm converges at different speeds. On the one hand, if s_j is small, then $(1-\eta s_j^2/N)$ is close to 1, which leads to a slow convergence. On the other hand, for large s_j, $(1-\eta s_j^2/N)$ is close to 0 and the convergence is very fast.

Furthermore, for $\boldsymbol{\omega}(n) = \mathbf{P}\mathbf{b}(n)$, we have

$$\boldsymbol{\omega}_{KA,n} = \mathbf{P}\text{diag}\left(\left[1-\left(1-\eta s_1^2/N\right)^n\right]s_1^{-1},\ldots,\left[1-\left(1-\eta s_r^2/N\right)^n\right]s_r^{-1},0,\ldots,0\right)\mathbf{Q}^T\mathbf{d} \tag{2.63}$$

It means the reg-function for the kernel ADALINE stopped at iteration n is

$$H_{KA,n}(x) = 1-\left(1-\eta x^2/N\right)^n$$

which is similar to equation (2.50) for KLMS except the exponent. In the following theorem, we explicitly establish an upper bound for the solution norm $\|\boldsymbol{\omega}_{KA,n}\|$.

Lemma 2.5. *Assume* $|1 - \eta x^2/N| < 1$ *and* $x \geq 0$.

$$\left|\frac{1-\left(1-\eta x^2/N\right)^n}{x}\right| \leq \sqrt{\frac{2\eta}{N}}n$$

Proof. Let $z = \sqrt{\frac{\eta}{N}}x$ and $H(z) = \frac{1-\left(1-z^2\right)^n}{z}$.

$$\begin{aligned}
|H(z)| &= \frac{1}{z}\left[1-\left(1-z^2\right)\right]\left[1+\left(1-z^2\right)+\ldots+\left(1-z^2\right)^{n-1}\right] \\
&= z\left|\left[1+\left(1-z^2\right)+\ldots+\left(1-z^2\right)^{n-1}\right]\right| \\
&\leq z\left[1+\left|\left(1-z^2\right)\right|+\ldots+\left|\left(1-z^2\right)^{n-1}\right|\right] \\
&\leq zn
\end{aligned}$$

for all z. Substituting $z = \sqrt{\frac{\eta}{N}}x$, we have

$$\left|\frac{1-\left(1-\eta x^2/N\right)^n}{x}\right| \leq \frac{\eta n}{N}x$$

Using the fact that $0 \leq x \leq \sqrt{2N/\eta}$, we have

$$\left|\frac{1-\left(1-\eta x^2/N\right)^n}{x}\right| \leq \sqrt{\frac{2\eta}{N}}n$$

□

Theorem 2.6. *Assume* $|1-\eta s_i^2/N|<1$, $\forall i$.

$$\|\boldsymbol{\omega}_{KA,n}\| \le \sqrt{\frac{2\eta}{N}} n \|\mathbf{d}\|$$

Proof.

$$\begin{aligned}\|\boldsymbol{\omega}_{KA,n}\| &= \left\|\mathbf{P}\,\mathrm{diag}\left(\left[1-\left(1-\eta s_1^2/N\right)^n\right]s_1^{-1},\ldots,\left[1-\left(1-\eta s_r^2/N\right)^n\right]s_r^{-1},0,\ldots,0\right)\mathbf{Q}^T\mathbf{d}\right\| \\ &\le \left\|\mathrm{diag}\left(\left[1-\left(1-\eta s_1^2/N\right)^n\right]s_1^{-1},\ldots,\left[1-\left(1-\eta s_r^2/N\right)^n\right]s_r^{-1},0,\ldots,0\right)\right\|\|\mathbf{d}\| \\ &\le \sqrt{\frac{2\eta}{N}} n \|\mathbf{d}\| \quad \text{(using Lemma 2.5)}\end{aligned}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are orthogonal matrices. □

On the one hand, the bound just derived reveals a great deal of insight into the adaptation. We note that small n (the number of iterations) or small η gives a smaller bound, which indicates more regularization. On the other hand, small N (the size of training data) makes the bound larger, which indicates less regularization.[5]

2.10 RESOURCE ALLOCATING NETWORKS

A resource-allocating network (RAN) described by Platt [1991] is probably the earliest attempt in this research direction. Although RAN is fundamentally different from kernel adaptive filters, its learning procedure bears some resemblance to KLMS. Also, many of our ideas are influenced directly by this pioneer algorithm such as the novelty criterion.

RAN is a growing radial-basis function network. It stores the centers, the widths of the centers, and the linear coefficients in the format of $\{\mathbf{c}_j, w_j, a_j\}$ for the jth unit. The calculation of the output for an input pattern $\mathbf{u}$ is given by

$$\begin{aligned}x_j &= \exp\left(-\|\mathbf{u}-\mathbf{c}_j\|^2/w_j^2\right) \\ y &= \sum_j a_j x_j + \gamma\end{aligned}$$

where γ is a bias term.

The learning strategy is as follows: The network starts with a blank slate. When $\{\mathbf{u}, d\}$ is identified as a pattern that is not currently well presented by the network, the network allocates a new unit that memorizes the pattern. Let the index of this new unit be n. The center of the unit is set to the novel input

$$\mathbf{c}_n = \mathbf{u}$$

The linear coefficient on the second layer is set to the difference between the output of the network and the novel output

$$a_n = d - y$$

The width of the new unit is proportional to the distance from the nearest stored center to the novel input

$$w_n = k \|\mathbf{u} - \mathbf{c}_{\text{nearest}}\|$$

where k is an overlap factor. As k grows larger, the responses of the units overlap more and more.

RAN uses a two-part novelty condition. An input–output pair $\{\mathbf{u}, d\}$ is considered novel if the input is far away from existing centers

$$\|\mathbf{u} - \mathbf{c}_{\text{nearest}}\| > \delta(t)$$

and if the difference between the desired output and the output of the network is large

$$\|d - y(\mathbf{u})\| > \delta_2$$

Errors larger than δ_2 are immediately corrected by the allocation of a new unit, whereas errors smaller than δ_2 are gradually repaired using gradient descent. The distance $\delta(t)$ is the scale of resolution that the network is fitting at the tth input presentation. The learning session starts with $\delta(t) = \delta_{\max}$, which is the largest length scale of interest, typically the size of the entire input space of nonzero probability density. The distance $\delta(t)$ shrinks until it reaches $\delta_{\min}$, which is the smallest length scale of interest. The following function is used to determine $\delta(t)$:

$$\delta(t) = \max(\delta_{\max} \exp(-t/\tau), \delta_{\min})$$

where τ is a decay constant.

When a new unit is not allocated, the LMS algorithm is used to decrease the error

$$\Delta a_j = \eta(d - y)x_j$$

$$\Delta\gamma = \eta(d - y)$$

$$\Delta\mathbf{c}_j = \frac{2\eta}{w_j}(\mathbf{u} - \mathbf{c}_j)x_j[(d - y)a_j]$$

It is shown that RAN can learn quickly and accurately, and it can form a compact representation. However, we have to point out that RAN is built on

intuition and heuristics; it is not a convex optimization problem, and its convergence is hard to prove and not guaranteed. Unlike RAN, KLMS is not restricted to the Gaussian kernel and uses a step-size parameter to correct the error gradually. On the whole, KLMS is conceptually and practically simpler.

2.11 COMPUTER EXPERIMENTS

2.11.1 KLMS Applied to Mackey–Glass Time-Series Prediction

The first example is the short-term prediction of the Mackey–Glass (MG) chaotic time series.[6] It is generated from the following time-delay ordinary differential equation:

$$\frac{dx(t)}{dt} = -bx(t) + \frac{ax(t-\tau)}{1+x(t-\tau)^{10}} \tag{2.64}$$

with $b = 0.1$, $a = 0.2$, and $\tau = 30$. The time series is discretized at a sampling period of 6 seconds. A segment of 5000 points of the time series is generated using equation (2.74) and stored in the mat file MK30.mat.

The first question is how to select the best filter order. Inspired by chaos theory, a principled approach to select the minimal filter order that preserves the shape of the trajectories (after the transients die down) is called the Takens embedding theorem [Takens, 1981]. According to this theorem, the optimal embedding for this system is around 7. In this example, we choose the time embedding as 10, i.e., $\mathbf{u}(i) = [x(i-10), x(i-9), \ldots, x(i-1)]^T$ (the 10 most recent values in the past) are used as the input to predict the present one $x(i)$, which is the desired response in this example. The code for the experiment can be found at http://www.cnel.ufl.edu/~weifeng/publication.htm. The readers are encouraged to play with all the parameters.

Part 1: A segment of 500 samples is used as the training data and another 100 as the test data. The data are corrupted by additive Gaussian noise with zero mean and 0.04 standard deviation. The purpose of the experiment is to compare the performance of a linear combiner trained with LMS and KLMS. The step-size parameter for LMS is 0.2. For KLMS, the Gaussian kernel [equation (1.24)] with $a = 1$ is chosen and the step-size parameter is also 0.2. Figure 2.5 is a typical plot of the learning curves. At each iteration, the MSE is computed on the test set using the filter resulting from in the training set. As expected, KLMS converges to a smaller value of MSE because of its nonlinear nature. Surprisingly, the rate of decay of both learning curves is basically the same, which suggests that the eigenvalue spread in the RKHS is similar to that of the input space.

Part 2: This is a more comprehensive comparison among LMS, KLMS, and a regularization network (RN), which serves as a batch-mode baseline. RN is a classic nonlinear modeling tool using a radial-basis function network topol-

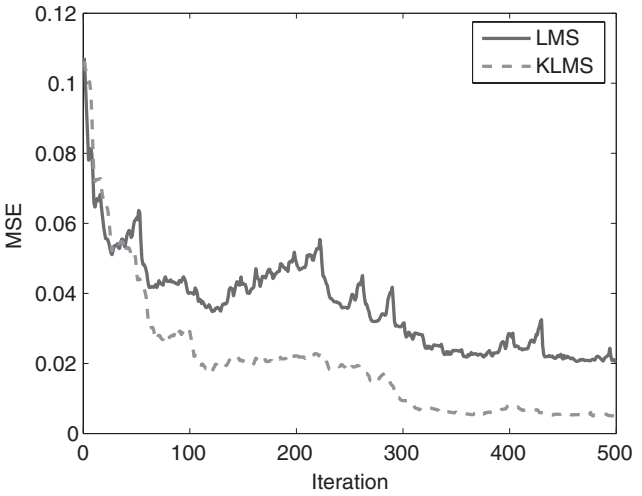


Figure 2.5. Learning curves of LMS and KLMS in Mackey–Glass time-series prediction.

Table 2.1. Performance comparison of KLMS with different step sizes and RN with different regularization parameters in Mackey–Glass time-series prediction.

Algorithm	Training MSE	Testing MSE
Linear LMS	0.021 ± 0.002	0.026 ± 0.007
KLMS ($\eta = 0.1$)	0.0074 ± 0.0003	0.0069 ± 0.0008
KLMS ($\eta = 0.2$)	0.0054 ± 0.0004	0.0056 ± 0.0008
KLMS ($\eta = 0.6$)	0.0062 ± 0.0012	0.0058 ± 0.0017
RN ($\lambda = 0$)	0 ± 0	0.012 ± 0.004
RN ($\lambda = 1$)	0.0038 ± 0.0002	0.0039 ± 0.0008
RN ($\lambda = 10$)	0.011 ± 0.0001	0.010 ± 0.0003

ogy specified by the kernel used [Poggio and Girosi, 1990]. The Gaussian kernel with $a = 1$ is chosen for both RN and KLMS. In RN, every input point is used as the center and the training is done in batch mode. One hundred Monte Carlo simulations are run with different realizations of noise. The results are summarized in Table 2.1.

All the results in these tables are in the form of "average ± standard deviation." As we can observe in Table 2.1, the performance of KLMS is much better than the linear LMS, which is to be expected (Mackey–Glass time series is a nonlinear system) and is comparable with RN with the best regularization. This is indeed surprising because RN can be viewed as a batch mode kernel regression method versus KLMS, which is a straight stochastic gradient approach implemented in RKHS. It is interesting to compare the design and performance of KLMS with different step sizes and RN with different regularization parameters, because each controls the stability of the obtained

Table 2.2. Complexity comparison of LMS, KLMS, and RN at iteration i.

Algorithm	Computation	Memory
LMS	$O(L)$	$O(L)$
KLMS	$O(i)$	$O(i)$
RN	$O(i^3)$	$O(i^2)$

Table 2.3. Solution norms of KLMS with different step sizes and RN with different regularization parameters in Mackey–Glass time-series prediction.

Algorithm	Solution norm
KLMS ($\eta = 0.1$)	0.84 ± 0.02
KLMS ($\eta = 0.2$)	1.14 ± 0.02
KLMS ($\eta = 0.6$)	1.73 ± 0.06
RN ($\lambda = 0$)	3375 ± 639
RN ($\lambda = 1$)	1.47 ± 0.03
RN ($\lambda = 10$)	0.55 ± 0.01

solution. First of all, when the regularization parameter is zero, the RN performs poorly on the test set (worse than the linear solution), which indicates that the solution is poorly regularized. RN is capable of outperforming KLMS with the proper regularization parameter ($\lambda = 1$), but the difference is small and at the expense of a more complex solution as well as with a careful selection of the regularization parameter.

Table 2.2 summarizes the computational complexity of the three algorithms. KLMS effectively reduces the computational complexity and memory storage when compared with RN.

Part 3: We compute the solution norms to support our theory that the norm of the KLMS solution is well bounded. As we observe in Tables 2.1 and 2.3, increasing the step-size parameter in KLMS increases the norm of the solution but fails to increase the performance because of the gradient noise in the estimation (misadjustment).

Part 4: Different noise variances σ^2 are used in the data to validate KLMS's applicability. As we observe in Tables 2.4 and 2.5, KLMS performs consistently on the training and test sets with different noise levels and degrades gracefully with increasing noise variance. It is observed that at severe noise level ($\sigma = .5$), all methods fall apart because the noise component will no longer correspond to the smallest singular value as required by Tikhonov regularization. With small noise, the regularization network outperforms KLMS because the misadjustment becomes the bottleneck. This is a good illustration of the difficulty KLMS may face to balance among convergence, misadjustment, and regularization. But remember, KLMS is a much simpler,

Table 2.4. Performance comparison of LMS, KLMS, and RN with different noise levels in Mackey–Glass time-series prediction (training MSE).

Algorithm	Linear LMS	KLMS ($\eta = 0.1$)	RN ($\lambda = 1$)
$\sigma = .005$	$0.017 \pm 5e-5$	$0.0050 \pm 2e-5$	$0.0014 \pm 1e-5$
$\sigma = .02$	0.018 ± 0.0002	0.0055 ± 0.0001	$0.0020 \pm 6e-5$
$\sigma = .04$	0.021 ± 0.002	0.0074 ± 0.0003	0.0038 ± 0.0002
$\sigma = .1$	0.033 ± 0.001	0.019 ± 0.001	0.010 ± 0.0005
$\sigma = .5$	0.326 ± 0.015	0.252 ± 0.010	0.097 ± 0.003

Table 2.5. Performance comparison of LMS, KLMS, and RN with different noise levels in Mackey–Glass time-series prediction (testing MSE).

Algorithm	Linear LMS	KLMS ($\eta = 0.1$)	RN ($\lambda = 1$)
$\sigma = .005$	0.018 ± 0.0002	0.0041 ± 0.0001	$0.0012 \pm 6e-5$
$\sigma = .02$	0.018 ± 0.0007	0.0046 ± 0.0004	0.0016 ± 0.0002
$\sigma = .04$	0.026 ± 0.007	0.0069 ± 0.0008	0.0039 ± 0.0008
$\sigma = .1$	0.031 ± 0.005	0.018 ± 0.003	0.017 ± 0.003
$\sigma = .5$	0.363 ± 0.057	0.332 ± 0.052	0.331 ± 0.052

Table 2.6. Effect of kernel size of Gaussian kernel on performance of KLMS and RN in Mackey–Glass time-series prediction.

Algorithm	Training MSE	Testing MSE
Linear LMS	0.022 ± 0.001	0.022 ± 0.001
KLMS ($a = 10$)	0.0085 ± 0.0005	0.0078 ± 0.0010
KLMS ($a = 2$)	0.0061 ± 0.0003	0.0056 ± 0.0014
KLMS ($a = .2$)	0.017 ± 0.0007	0.016 ± 0.0010
RN ($a = 10$)	0.0040 ± 0.0002	0.0068 ± 0.0009
RN ($a = 2$)	0.0043 ± 0.0002	0.0047 ± 0.0006
RN ($a = .2$)	0.0098 ± 0.0003	0.0092 ± 0.0005

online algorithm and the performance gap compared with RN is the price to be paid. Throughout this set of simulations, the kernel used in KLMS and RN is the Gaussian kernel with $a = 1$. The learning step is 0.1 for both the linear LMS and KLMS. The regularization parameter of RN is set at the best value ($\lambda = 1$).

Part 5: Any kernel method, including KLMS, needs to choose a suitable kernel and its bandwidth. The effect of different kernels and different kernel parameters on KLMS is demonstrated. In the case of the Gaussian kernel [equation (1.24)], we choose three kernel parameters: 10, 2, and 0.2. The learning rate is set at 0.1 for both the linear LMS and KLMS, and the regularization parameter of RN is 1 throughout the simulation. The results are summarized in Table 2.6. As expected, too small or too large kernel sizes hurt performance

Table 2.7. Effect of order of polynomial kernel on performance of KLMS and RN in Mackey-Glass time-series prediction.

Algorithm	Training MSE	Testing MSE
KLMS ($p = 2, \eta = 0.1$)	0.010 ± 0.001	0.009 ± 0.002
KLMS ($p = 5, \eta = 0.01$)	0.0099 ± 0.0006	0.0099 ± 0.0007
KLMS ($p = 8, \eta = .0006$)	0.027 ± 0.009	0.025 ± 0.009
RN ($p = 2, \lambda = 1$)	0.0064 ± 0.0005	0.0066 ± 0.0008
RN ($p = 5, \lambda = 1$)	0.0034 ± 0.0003	0.0059 ± 0.0007
RN ($p = 8, \lambda = 1$)	0.0014 ± 0.0001	0.0078 ± 0.0004

Table 2.8. Performance comparison of LMS and KLMS with different training data sizes.

Algorithm	Training MSE	Testing MSE
LMS ($N = 1000$)	0.020 ± 0.0004	0.019 ± 0.0015
LMS ($N = 2000$)	0.019 ± 0.0004	0.018 ± 0.0009
LMS ($N = 4000$)	0.019 ± 0.0003	0.020 ± 0.0016
KLMS ($N = 1000$)	0.0060 ± 0.0002	0.0062 ± 0.0009
KLMS ($N = 2000$)	0.0058 ± 0.0002	0.0053 ± 0.0010
KLMS ($N = 4000$)	0.0054 ± 0.0001	0.0058 ± 0.0007

for both KLMS and RN. In this problem, a kernel size around 1 gives the best performance on the test set.

Part 6: In the case of the polynomial kernel [equation (1.25)], the order is set to 2, 5, and 8. The learning rate is chosen accordingly in KLMS as listed in Table 2.7 (recall the relation between the learning rate and the trace of the Gram matrix). It is observed that the performance deteriorates substantially when p is too large (>8) for KLMS. This is also validated by the misadjustment formula [equation (2.28)].

Part 7: It is noted in the theoretical analysis that the training data size will not affect the regularization of KLMS. To illustrate this behavior, we choose different training data sizes to observe how KLMS performs. The noise variance is set at 0.05, and the numbers of training data are 1000, 2000, and 4000, respectively. Other parameters are the same as in the first set of simulations. As presented in Table 2.8, KLMS performs consistently on the training and test sets with an increasing number of training data.

Part 8: In this simulation, we examine the effect of the regularization parameter on the performance of NORMA (leaky KLMS). Twenty regularization parameters are chosen within [0, 0.1]. For each regularization parameter, 50 Monte Carlo simulations are performed with different realizations of noise ($\sigma = 0.01$). The final average MSE on the testing set is plotted in Figure 2.6 along with its standard deviation. As we can observe, the explicit regularization has a detrimental effect in this example.

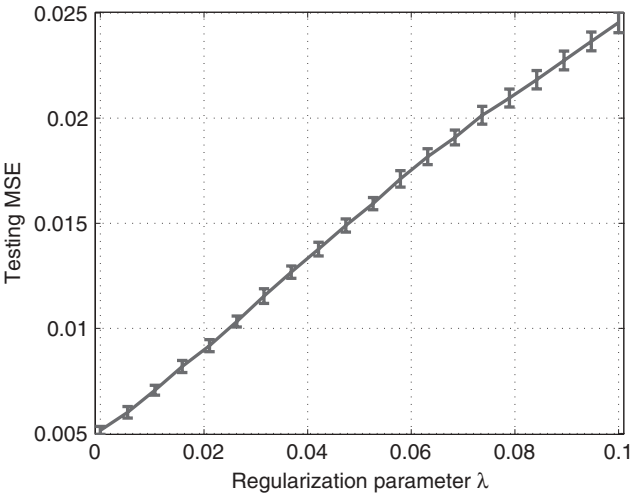


Figure 2.6. Performance of NORMA with explicit regularization in Mackey–Glass time-series prediction.

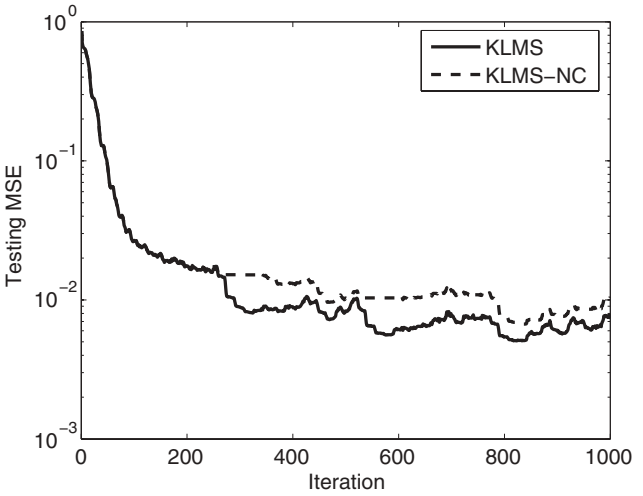


Figure 2.7. Learning curves of KLMS with and without novelty criterion in Mackey–Glass time-series prediction.

Part 9: We next test how the novelty criterion affects the performance. A segment of 1000 samples is used as the training data and another 200 as the test data. All the data are corrupted by Gaussian noise with zero mean and 0.0001 variance. The typical learning curves are shown in Figure 2.7 with the thresholds in the novelty criterion $\delta_1 = 0.1$ and $\delta_2 = 0.05$. The step-size parameter used is 0.1.With the previous results, we know that the optimal kernel

Table 2.9. Performance of KLMS with novelty criterion in Mackey–Glass time-series prediction.

Algorithm	Parameters	Testing MSE	Dictionary size
KLMS		0.0062 ± 0.00048	1000
KLMS-NC	$\delta_1 = 0.05$, $\delta_2 = 0.02$	0.0065 ± 0.00051	754
KLMS-NC	$\delta_1 = 0.05$, $\delta_2 = 0.05$	0.0066 ± 0.00048	528
KLMS-NC	$\delta_1 = 0.05$, $\delta_2 = 0.1$	0.0072 ± 0.00033	286
KLMS-NC	$\delta_1 = 0.1$, $\delta_2 = 0.05$	0.0078 ± 0.00055	490
KLMS-NC	$\delta_1 = 0.2$, $\delta_2 = 0.05$	0.0134 ± 0.00041	284

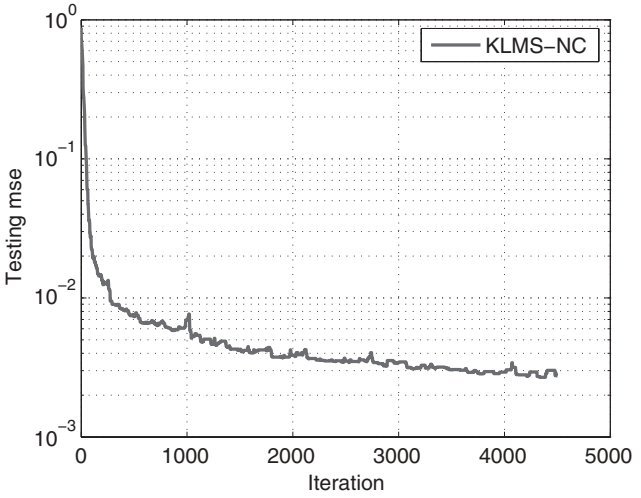


Figure 2.8. Learning curve of KLMS with novelty criterion in Mackey–Glass time-series prediction.

bandwidth is around 1, which means that δ_1 is about 0.07 ($0.1/\sqrt{2a}$). Also, the testing MSE is around 0.006, which means δ_2 is about 0.08 ($\sqrt{0.006}$). Different thresholds are tested, and the results are summarized in Table 2.9. The MSE is calculated from the last 100 points of the learning curves. It is observed that the complexity can be reduced dramatically with the novelty criterion to preserve the prediction accuracy. Of course, with δ_1 and δ_2 too large, the performance degrades.

Part 10: We examine how the novelty criterion affects the growth pattern of KLMS. A segment of 4500 samples is used as the training data and another 200 as the test data. All the data are corrupted by Gaussian noise with zero mean and 0.0001 variance. The thresholds in the novelty criterion are set as $\delta_1 = 0.05$ and $\delta_2 = 0.1$. All other settings are the same as in **Part 9**. The learning curve is plotted in Figure 2.8. The growth curve in Figure 2.9 shows the network size at each iteration. Only 571 inputs out of 4500 (13%) are eventu-

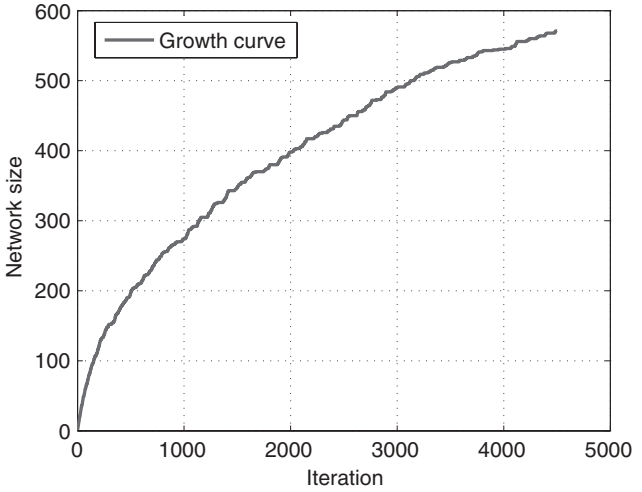


Figure 2.9. Growth curve of KLMS with novelty criterion in Mackey–Glass time-series prediction.

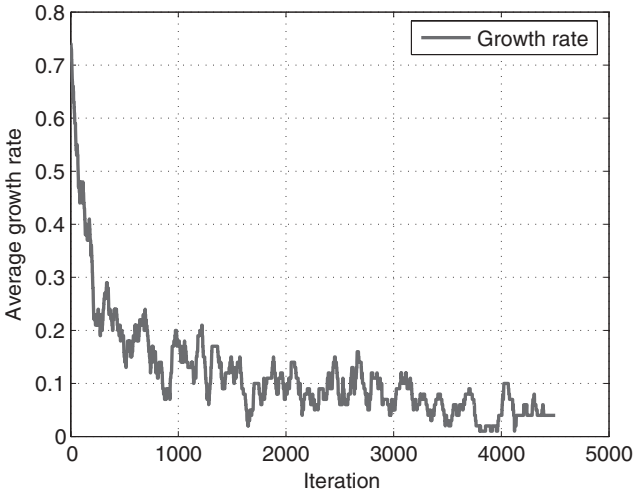


Figure 2.10. Average growth rate curve of KLMS with novelty criterion in Mackey–Glass time-series prediction.

ally selected into the dictionary. The growth rate curve shows the average growth rate in a fixed-width window (window length is 100 in Figure 2.10). It is observed that the network growth is effectively contained with the novelty criterion. The growth rate drops dramatically from around 0.8 to 0.05. This is perhaps the worst-case scenario because the Mackey–Glass time series is chaotic and never repeats itself.

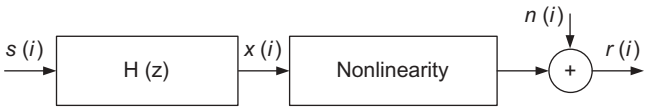


Figure 2.11. Basic structure of a nonlinear channel.

2.11.2 KLMS Applied to Nonlinear Channel Equalization

The LMS algorithm is used widely in channel equalization, and we tested KLMS on a nonlinear channel equalization problem.[7] The nonlinear channel model consists of a serial connection of a linear filter and a memoryless nonlinearity (see Figure 2.11). This kind of model has been used to model digital satellite communication channels and digital magnetic recording channels.

The problem setting is as follows: A binary signal $\{s(1), s(2), \ldots, s(N)\}$ is fed into a nonlinear channel. At the receiver end of the channel, the signal is corrupted by additive white Gaussian noise and is then observed as $\{r(1), r(2), \ldots, r(N)\}$. The aim of channel equalization is to construct an "inverse" filter that reproduces the original signal with as low an error rate as possible. It is easy to formulate it as a regression problem, with examples $\{([r(i), r(i+1), \ldots, r(i+l)], s(i-D))\}$, where l is the time-embedding length and D is the equalization time lag. $l = 5$ and $D = 2$ are used in this experiment. The nonlinear channel model is defined by $x(i) = s(i) + 0.5s(i-1)$, $r(i) = x(i) - 0.9x(i)^2 + n(i)$, where $n(i)$ is the white Gaussian noise with a variance of σ^2.

Part 1: The performance of LMS, KLMS, and RN (the latter as a batch-mode baseline) are compared. The filters are trained with 1000 data and fixed afterward. Testing is performed on a 5000-sample random test sequence. The Gaussian kernel with $a = 0.1$ and step-size parameter $\eta = 0.2$ are used in KLMS for best results. The step-size parameter of LMS is set at 0.01. Figure 2.12 is a typical plot of the learning curves.

Part 2: The three algorithms, namely LMS, KLMS, and RN, are tested on this problem with different noise levels. The results are presented in Table 2.10; each entry consists of the average and the standard deviation for 100 Monte Carlo independent tests. The results show that RN outperforms KLMS in terms of bit error rate (BER) but not by much, which is surprising because one is a batch method and the other is online. They both outperform the conventional LMS substantially as can be expected because the channel is nonlinear.

2.12 CONCLUSION

The KLMS algorithm is a stochastic gradient methodology to solve least-squares problems in RKHS. Because the update equation can be written in terms of inner product, KLMS can be computed efficiently in the input space. The good approximation ability of KLMS stems from the fact that the transformed data include possibly infinite different features of the original data. In

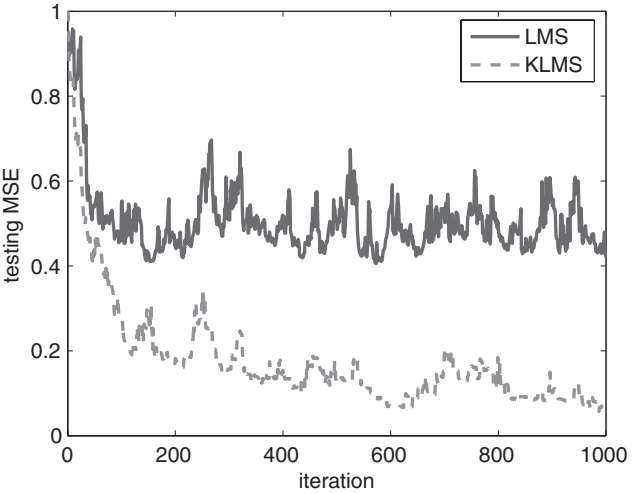


Figure 2.12. Learning curves of LMS and KLMS in nonlinear channel equalization ($\sigma = 0.4$).

Table 2.10. Performance comparison of LMS, KLMS, and RN in nonlinear channel equalization.

Algorithm	Linear LMS ($\eta = .005$)	KLMS ($\eta = 0.1$)	RN ($\lambda = 1$)
BER ($\sigma = .1$)	0.162 ± 0.014	0.020 ± 0.012	0.008 ± 0.001
BER ($\sigma = .4$)	0.177 ± 0.012	0.058 ± 0.008	0.046 ± 0.003
BER ($\sigma = .8$)	0.218 ± 0.012	0.130 ± 0.010	0.118 ± 0.004

the framework of stochastic projection, the space spanned by $\{\boldsymbol{\varphi}(i)\}$ is so large that the projection error of the desired signal $d(i)$ could be small [Parzen, 1959], as is well known from Cover's theorem [Haykin, 2009]. This capability includes modeling of nonlinear systems, which is the main reason why KLMS can achieve good performance in the Mackey–Glass system prediction and nonlinear channel equalization.

As demonstrated by the experiments, KLMS has general applicability because of its simplicity; in particular, it does not need to work with large Gram matrices as most of the kernel algorithms because it uses the data on the basis of one sample at a time. KLMS may be useful in problems like nonlinear channel equalization, nonlinear system identification, and nonlinear active noise control, where online filters are a necessity. Almost all the literature for LMS can be used to analyze KLMS; in particular, its convergence and stability are well understood. Also in the framework of RKHS, any Mercer kernel can be used in KLMS instead of restricting the architecture to the Gaussian kernel as in RAN.

KLMS is a simple and effective nonlinear filter design. It has the universal approximation capability in stationary environments. Its convergence property and regularization property are controlled mainly by a simple parameter (step-size parameter). Practical approaches are available to select the kernel, to choose the step-size parameter, and to contain the network growth. Issues that require future investigation include pruning methods to reduce network size and adaptive mechanisms for kernel size to capture local data structure more accurately.

ENDNOTES

1. **Radial-Basis Function Networks.** Radial-basis function networks are motivated to find a surface in a multidimensional space that provides a best fit to the training data, with the criterion for "best fit" being measured in some statistical sense. They were first introduced in the solution of the real multivariate interpolation problem. The early work on this subject is surveyed in Powell [1985]. A survey of their use in the field of neural networks can be found in Light [1992].

   In a strict sense, the interpolation problem may be stated:

   Given a set of N different points $\{\mathbf{x}_i \in \mathbb{R}^L\}_{i=1}^N$ and a corresponding set of N real numbers $\{d_i \in \mathbb{R}\}_{i=1}^N$, find a function $f: \mathbb{R}^L \to \mathbb{R}$ that satisfies the interpolation condition

   $$f(\mathbf{x}_i) = d_i, \quad i = 1, 2, \ldots, N \tag{2.65}$$

   For strict interpolation as specified above, the interpolating surface is constrained to pass through all the training data points, which may be undesirable when the observed data are noisy. The RBF technique chooses a function f, which is a linear combination of a set of basis functions:

   $$f(\mathbf{x}) = \sum_{i=1}^{N} a_i g(\|\mathbf{x} - \mathbf{x}_i\|) \tag{2.66}$$

   where $g(\|\mathbf{x} - \mathbf{x}_i\|)$ is a set of N arbitrary functions, which are known as radial-basis functions, and $\|\cdot\|$ denotes a norm between $\mathbf{x} - \mathbf{x}_i$, which is usually Euclidean. Notice that the centers of the radial-basis functions are the regressors from the observed data. Using the condition of interpolation of equation (2.65) in equation (2.66), we have

   $$\begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1N} \\ g_{21} & g_{22} & \cdots & g_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ g_{N1} & g_{N2} & \cdots & g_{NN} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \cdots \\ a_N \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \cdots \\ d_N \end{pmatrix} \tag{2.67}$$

   where

   $$g_{ij} = g(\|\mathbf{x}_i - \mathbf{x}_j\|), \quad i = 1, \ldots, N; \; j = 1, \ldots, N$$

If the function form of g is fixed and known, we can solve this linear system for the unknown coefficients $\{a_i\}_{i=1}^N$. Let $\mathbf{G}$ denote an $N \times N$ matrix with elements g_{ij} at the (i, j)th entry and

$$\mathbf{d} = [d_1, d_2, \ldots, d_N]^T$$
$$\mathbf{a} = [a_1, a_2, \ldots, a_N]^T$$

$\mathbf{G}$ is called the *interpolation matrix*, $\mathbf{d}$ the *desired response vector*, and $\mathbf{a}$ the *linear weight vector*. We may rewrite (2.67) in the compact form

$$\mathbf{G}\mathbf{a} = \mathbf{d} \tag{2.68}$$

By the matrix theory, we know that a unique solution of $\mathbf{a}$ exists, if and only if $\mathbf{G}$ is invertible. An important theorem proved by Micchelli [1986] states:

> Let $\{\mathbf{x}_i\}_{i=1}^N$ be a set of *distinct* points in $\mathbb{R}^L$ and g an arbitrary nonlinear function. Then the N-by-N interpolation matrix $\mathbf{G}$, whose ijth element is $g_{ij} = g(\|\mathbf{x}_i - \mathbf{x}_j\|)$, is *nonsingular*.

Therefore, as long as the inputs are distinct, the inverse of $\mathbf{G}$ exists and the linear weight vector can be simply solved by

$$\mathbf{a} = \mathbf{G}^{-1}\mathbf{d} \tag{2.69}$$

A large class of radial-basis functions is covered by Micchelli's theorem. Commonly used types of radial basis functions include the following:

1. Gaussian:

$$g(\|\mathbf{x}_i - \mathbf{x}_j\|) = \exp\left(-a\|\mathbf{x}_i - \mathbf{x}_j\|^2\right) \text{ for some } a > 0 \tag{2.70}$$

2. Multiquadrics:

$$g(\|\mathbf{x}_i - \mathbf{x}_j\|) = \sqrt{\|\mathbf{x}_i - \mathbf{x}_j\|^2 + c^2} \text{ for some } c > 0 \tag{2.71}$$

3. Inverse multiquadric:

$$g(\|\mathbf{x}_i - \mathbf{x}_j\|) = \frac{1}{\sqrt{\|\mathbf{x}_i - \mathbf{x}_j\|^2 + c^2}} \text{ for some } c > 0 \tag{2.72}$$

4. Thin plate spline:

$$g(\|\mathbf{x}_i - \mathbf{x}_j\|) = \|\mathbf{x}_i - \mathbf{x}_j\|^2 \ln(\|\mathbf{x}_i - \mathbf{x}_j\|) \tag{2.73}$$

Clearly, RBF networks, in the form of equation (2.66), have the same *shallow* structure as KLMS. The linear coefficients are solved by inverting the interpolation

matrix. The complexity in the training phase follows a cubic rule $O(N^3)$. Another important thing is that the interpolation matrix is not guaranteed to be positive definite. For example, an interpolation matrix made from the multiquadrics of equation (2.71) has $N-1$ negative eigenvalues and only one positive eigenvalue [Micchelli, 1986].

2. **Kernel Selection.** For a thorough treatment on reproducing kernels, see Schölkopf and Smola [2002], Shawe-Taylor and Cristianini [2004], and Rasmussen and Williams [2006].

   All the kernel methods need to choose the kernel type and parameters. The most popular method so far is by cross-validation [Racine, 1993, Cawley and Talbot, 2003, An et al., 2007]. The nearest-neighbor method is also used in the resource-allocating networks that allow the adaptation of the kernel size during the learning. With a close relation to Gaussian process theory, maximum marginal likelihood [Rasmussen and Williams, 2006] is also applicable, which we will discuss later. In addition, the general kernel selection problem has been studied as a convex optimization through parameterization of the kernel function family [Micchelli and Pontil, 2005, Argyriou et al., 2005, Chapelle et al., 2002].

3. **Pruning in Kernel Methods.** Simple pruning strategies include pruning the oldest unit in the dictionary [Van Vaerenbergh et al., 2006], pruning randomly [Cavallanti et al., 2007], pruning the unit with the least coefficient [Dekel et al., 2006], and pruning the unit with the smallest outputs on recent inputs.

4. **Low-Rank Approximation.** Because the kernel space is a high-dimensional space (can be infinite in the case of a Gaussian kernel), to solve the adaptive filtering problem directly in the primal space requires low-rank approximation methods such as the Nyström method [Williams and Seeger, 2001], incomplete Cholesky factorization [Fine and Scheinberg, 2001], and kernel principal component analysis [Schölkopf et al., 1998].

5. **Gradient Descent is Regularization.** The gradient descent method is known for its regularization property in the literature as illustrated by the deterministic analysis of early stopping in inverse problems (see Raudys and Cibas, 1996, Engl et al., 2000, Hagiwara and Kuno, 2000, Yao et al., 2007).

6. **Mackey–Glass Equation.** The Mackey–Glass equation is a nonlinear time-delay differential equation

$$\frac{dx(t)}{dt} = -bx(t) + \frac{ax(t-\tau)}{1+x(t-\tau)^n} \tag{2.74}$$

   where a, b, n, and τ are real numbers. This equation displays characteristics of periodic and chaotic dynamics. Mackey and Glass [1977] first used it to model physiological control systems such as electrolytes, oxygen, glucose, and blood cells in the blood, blood pressure to the brain and various organs. Equation (2.74) represents a typical feedback system. In real feedback systems, there is typically a time lag τ between the sensing of the value of a variable under control and the mounting of an appropriate response, which requires the dependency of x on the time-delayed value of $x(t-\tau)$. For example, after a loss of blood cells, it can take many days before new blood cells can be produced through the activation, differentiation, and proliferation of the appropriate blood stem cells. Then, Farmer [1982] recognized

that increasing the value of τ in equation (2.74) increases the dimension of the attractor in this chaotic system. This observation, and the simplicity of the equation, has led to the evolution of this equation into a standard model used to test algorithms for nonlinear modeling capability [Farmer and Sidorowich, 1987, Crowder, 1990, Platt, 1991, Martinetz et al., 1993, Mukherjee et al., 1997, Müller et al., 1997]. For more details, please refer to Glass and Mackey [1988] and Beuter et al. [2003].

7. **Adaptive Channel Equalization.** In 1965, Lucky [1965] made a major breakthrough in the equalization problem by proposing a *zero-forcing algorithm* for automatically adjusting the tap weights of a transversal equalizer. Gersho [1969] and Proakis and Miller [1969] independently reformulated the adaptive equalization problem using a mean-square-error criterion. In 1972, using the LMS algorithm, Ungerboeck presented a detailed mathematical analysis of the convergence properties of an adaptive transversal equalizer. In 1974, Godard used Kalman filter theory to derive a powerful algorithm for adjusting the tap weights of a transversal equalizer.

   It has been shown by Sayed [2003] that the optimal equalizer for a linear channel is actually nonlinear. Also from the viewpoint of communication theory, any physical channel exhibits nonlinear characteristics to some extent [Proakis, 2000]. Most notable examples include digital satellite communication channels [Benedetto and Biglieri, 1983, Kechriotis et al., 1994] and digital magnetic recording channels [Sands and Cioffi, 1993]. Theodoridis et al. [1992] presented a review of the use of clustering techniques for the channel equalization problem. The application of a radial-basis function network to digital communications channel equalization was examined in Chen et al. [1993a,b]. It is shown that the radial-basis function network can be employed to implement the optimal Bayesian symbol-decision equalizer. Cha and Kassam [1995] investigated the use of a complex-value radial-basis function network. Kechriotis et al. [1994] introduced an adaptive recurrent neural network (RNN)-based equalizer, which is suitable for high-speed channel equalization. RNN equalizers have comparable performance with traditional linear filter based equalizers when the channel interferences are relatively mild; however, they outperform the linear counterparts by several orders of magnitude when either the channel's transfer function has spectral nulls or severe nonlinear distortion is present. In addition, the small-size RNN equalizers are reported to outperform multilayer perceptron equalizers in many cases. Adali et al. [1997] have shown that the single and multilayer perceptron models can be used to implement the so-called maximum partial likelihood estimation that are very useful for dependent observations and sequential processing. More recently, support vector machines have been used to solve the equalization problem [Sebald and Bucklew, 2000]. Erdogmus et al. [2001] studied the use of multilayer perceptron equipped with information-theoretic cost functions to compensate the nonlinear effects caused by practical transmitter power amplifiers.

3

KERNEL AFFINE PROJECTION ALGORITHMS

This chapter extends the kernel least-mean-square (KLMS) algorithm to the class of algorithms that fall under Goodwin's online learning model, creating a rich, flexible, and cohesive taxonomy of online algorithms in reproducing kernel Hilbert Spaces (RKHS). The center piece of Goodwin's family is the affine projection algorithms (APAs), which inherit the simplicity and online nature of LMS while reducing the gradient noise by using multiple samples, therefore boosting LMS performance. APAs appear as intermediate complexity algorithms between the LMS and the recursive least squares (RLS).

As can be expected, the affine projection algorithms can be extended to RKHS using the basic methodology outlined for KLMS and give rise to the kernel affine projection algorithms (KAPA) family [Liu and Príncipe, 2008b]. Besides the number of samples, the other two degrees of freedom in the taxonomy are 1) the regularization in the cost function for better generalization and 2) the Newton updates that avoid the slowness of gradient descent produced by the eigenvalue spread of the input correlation matrix. Of course, the performance and computational complexity of all these versions are different, but they provide a full range of options to users trying to meet trade-offs between data rates (or database sizes) and hardware constraints.

More interestingly, KAPA provides a unifying model for several existing neural network techniques, including kernel least-mean-square algorithms, sliding-window kernel recursive least-squares algorithms, and regularization networks (Figure 1.5). Therefore, many insights can be gained into the basic relations among them and the trade-off between computation complexity and

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

performance. We will start with a review of affine projection algorithms, focusing on its subtle variations based on different optimization techniques. Then the matrix inversion lemma is used to derive equivalent representations that are more suitable for kernel extensions. Finally, the kernel affine projection algorithms follow naturally.

3.1 AFFINE PROJECTION ALGORITHMS

Let d be a zero-mean scalar-valued random variable, and let $\mathbf{u}$ be a zero-mean $L \times 1$ random variable with a positive-definite covariance matrix $\mathbf{R}_\mathbf{u} = \mathbf{E}[\mathbf{u}\mathbf{u}^T]$. The cross-covariance vector of d and $\mathbf{u}$ is denoted by $\mathbf{r}_{d\mathbf{u}} = \mathbf{E}[d\mathbf{u}]$. The weight vector $\mathbf{w}$ that solves

$$\min_{\mathbf{w}} J(\mathbf{w}) = \mathbf{E}\left|d - \mathbf{w}^T\mathbf{u}\right|^2 \tag{3.1}$$

is given by $\mathbf{w}^o = \mathbf{R}_\mathbf{u}^{-1}\mathbf{r}_{d\mathbf{u}}$ (the Wiener solution) [Haykin, 2002].

Several methods to approximate $\mathbf{w}^o$ iteratively exist. For example, the gradient descent method:

$$\mathbf{w}(0) = \text{initial guess}; \quad \mathbf{w}(i) = \mathbf{w}(i-1) + \eta[\mathbf{r}_{d\mathbf{u}} - \mathbf{R}_\mathbf{u}\mathbf{w}(i-1)] \tag{3.2}$$

or the smoothed Newton's recursion to increase convergence speed:

$$\mathbf{w}(0) = \text{initial guess}; \quad \mathbf{w}(i) = \mathbf{w}(i-1) + \eta(\mathbf{R}_\mathbf{u} + \varepsilon\mathbf{I})^{-1}[\mathbf{r}_{d\mathbf{u}} - \mathbf{R}_\mathbf{u}\mathbf{w}(i-1)] \tag{3.3}$$

where ε is a small positive smoothing factor to prevent division-by-zero and η is the step-size parameter specified by the designer.

Stochastic-gradient algorithms replace the covariance matrix and the cross-covariance vector at each iteration by local data approximations. There are several ways for obtaining such approximations, the trade-off being computation complexity, convergence performance, and steady-state behavior. Assume that we have access to observations of the random variables d and $\mathbf{u}$ over time

$$\{d(1), d(2), \ldots\} \quad \text{and} \quad \{\mathbf{u}(1), \mathbf{u}(2), \ldots\}$$

The LMS algorithm simply uses the instantaneous values to approximate $\hat{\mathbf{R}}_\mathbf{u} = \mathbf{u}(i)\mathbf{u}(i)^T$ and $\hat{\mathbf{r}}_{d\mathbf{u}} = d(i)\mathbf{u}(i)$. The corresponding steepest-descent recursion [equation (3.2)] and Newton's recursion [equation (3.3)] become

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta\mathbf{u}(i)\left[d(i) - \mathbf{u}(i)^T\mathbf{w}(i-1)\right] \tag{3.4}$$

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta\mathbf{u}(i)\left[\mathbf{u}(i)^T\mathbf{u}(i) + \varepsilon\mathbf{I}\right]^{-1}\left[d(i) - \mathbf{u}(i)^T\mathbf{w}(i-1)\right] \tag{3.5}$$

The affine projection algorithm, however, employs better approximations. Specifically, $\mathbf{R}_{\mathbf{u}}$ and $\mathbf{r}_{d\mathbf{u}}$ are replaced by the approximations from the K most recent inputs and observations. Denoting

$$\mathbf{U}(i) = [u(i-K+1), \ldots, \mathbf{u}(i)]_{L\times K} \quad \text{and} \quad \mathbf{d}(i) = [d(i-K+1), \ldots, d(i)]^T$$

we have

$$\begin{aligned} \hat{\mathbf{R}}_{\mathbf{u}} &= \frac{1}{K}\mathbf{U}(i)\mathbf{U}(i)^T \\ \hat{\mathbf{r}}_{d\mathbf{u}} &= \frac{1}{K}\mathbf{U}(i)\mathbf{d}(i) \end{aligned} \tag{3.6}$$

Therefore, equations (3.2) and (3.3) become

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta\mathbf{U}(i)\left[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)\right] \tag{3.7}$$

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta\left[\mathbf{U}(i)\mathbf{U}(i)^T + \varepsilon\mathbf{I}\right]^{-1}\mathbf{U}(i)\left[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)\right] \tag{3.8}$$

Notice that

$$\left[\mathbf{U}(i)\mathbf{U}(i)^T + \varepsilon\mathbf{I}\right]^{-1}\mathbf{U}(i) = \mathbf{U}(i)\left[\mathbf{U}(i)^T\mathbf{U}(i) + \varepsilon\mathbf{I}\right]^{-1}$$

This equation can be established by the matrix inversion lemma

$$(\mathbf{A}+\mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{C}^{-1} + \mathbf{D}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{D}\mathbf{A}^{-1} \tag{3.9}$$

with the identifications

$$\varepsilon\mathbf{I} \to \mathbf{A},\ \mathbf{U}(i) \to \mathbf{B},\ \mathbf{I} \to \mathbf{C},\ \mathbf{U}(i)^T \to \mathbf{D}$$

Therefore, equation (3.8) is equivalent to

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta\mathbf{U}(i)\left[\mathbf{U}(i)^T\mathbf{U}(i) + \varepsilon\mathbf{I}\right]^{-1}\left[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)\right] \tag{3.10}$$

It is noted that this equivalence deals with the matrix $[\mathbf{U}(i)^T\mathbf{U}(i) + \varepsilon\mathbf{I}]$ instead of $[\mathbf{U}(i)\mathbf{U}(i)^T + \varepsilon\mathbf{I}]$, and it plays an important role in the derivation of kernel extensions. We call recursion equation (3.7) APA-1 and recursion equation (3.10) APA-2. In the classic adaptive filtering literature, the name affine projection algorithm is exclusively used for the recursion equation (3.10), whereas we use affine projection algorithms to refer to a family of similar algorithms.[1]

In some circumstances, a regularized cost function is needed instead of equation (3.1). The regularized least-squares (LS) problem is

$$\min_{\mathbf{w}} \mathbf{E}\left|d - \mathbf{w}^T\mathbf{u}\right|^2 + \lambda\|\mathbf{w}\|^2 \tag{3.11}$$

where λ is the regularization parameter (do not confuse with the smoothing factor ε in Newton's recursion, which is introduced mainly to ensure numerical stability and is not directly related to the norm constraint implemented by λ). The gradient method for this new cost function becomes

$$\begin{aligned}\mathbf{w}(i) &= \mathbf{w}(i-1) + \eta[\mathbf{r}_{d\mathbf{u}} - (\lambda\mathbf{I} + \mathbf{R}_{\mathbf{u}})w(i-1)] \\ &= (1-\eta\lambda)\mathbf{w}(i-1) + \eta[\mathbf{r}_{d\mathbf{u}} - \mathbf{R}_{\mathbf{u}}\mathbf{w}(i-1)]\end{aligned} \tag{3.12}$$

and the Newton's recursion with $\varepsilon = 0$ is

$$\begin{aligned}\mathbf{w}(i) &= \mathbf{w}(i-1) + \eta(\lambda\mathbf{I} + \mathbf{R}_{\mathbf{u}})^{-1}[\mathbf{r}_{d\mathbf{u}} - (\lambda\mathbf{I} + \mathbf{R}_{\mathbf{u}})\mathbf{w}(i-1)] \\ &= (1-\eta)\mathbf{w}(i-1) + \eta(\lambda\mathbf{I} + \mathbf{R}_{\mathbf{u}})^{-1}\mathbf{r}_{d\mathbf{u}}\end{aligned} \tag{3.13}$$

If the approximations in equation (3.6) are used in equations (3.12) and (3.13), we have

$$\mathbf{w}(i) = (1-\eta\lambda)\mathbf{w}(i-1) + \eta\mathbf{U}(i)\left[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)\right] \tag{3.14}$$

and

$$\mathbf{w}(i) = (1-\eta)\mathbf{w}(i-1) + \eta\left[\lambda\mathbf{I} + \mathbf{U}(i)\mathbf{U}(i)^T\right]^{-1}\mathbf{U}(i)\mathbf{d}(i) \tag{3.15}$$

The latter, by the matrix inversion lemma, is equivalent to

$$\mathbf{w}(i) = (1-\eta)\mathbf{w}(i-1) + \eta\mathbf{U}(i)\left[\lambda\mathbf{I} + \mathbf{U}(i)^T\mathbf{U}(i)\right]^{-1}\mathbf{d}(i) \tag{3.16}$$

For simplicity, recursion in equations (3.14) and (3.16) are named APA-3 and APA-4, respectively.[2]

3.2 KERNEL AFFINE PROJECTION ALGORITHMS

Following the KLMS approach, the Mercer theorem is used to transform the data $\mathbf{u}(i)$ into the feature space $\mathbb{F}$ as $\boldsymbol{\varphi}(\mathbf{u}(i))$ [denoted as $\boldsymbol{\varphi}(i)$]. The affine projection algorithms are formulated on the example sequence $\{d(1), d(2), \ldots\}$ and $\{\boldsymbol{\varphi}(1), \boldsymbol{\varphi}(2), \ldots\}$ to estimate the weight vector $\boldsymbol{\omega}$ that solves

$$\min_{\boldsymbol{\omega}} E\left|d - \boldsymbol{\omega}^T\boldsymbol{\varphi}(\mathbf{u})\right|^2 \tag{3.17}$$

By straightforward manipulation, the stochastic gradient descent [equation (3.7)] becomes

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)\left[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)\right] \tag{3.18}$$

and stochastic Newton's method [equation (3.10)] becomes

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)\left[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \varepsilon\mathbf{I}\right]^{-1}\left[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)\right] \quad (3.19)$$

where $\boldsymbol{\Phi}(i) = [\boldsymbol{\varphi}(i-K+1), \ldots, \boldsymbol{\varphi}(i)]$.

Likewise, if the regularized cost function is specified, then equation (3.14) for the stochastic gradient descent becomes

$$\boldsymbol{\omega}(i) = (1-\lambda\eta)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)\left[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)\right] \quad (3.20)$$

and the corresponding Newton's method [equation (3.16)] becomes

$$\boldsymbol{\omega}(i) = (1-\eta)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)\left[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \lambda\mathbf{I}\right]^{-1}\mathbf{d}(i) \quad (3.21)$$

For simplicity, we refer to the recursions in equations (3.18), (3.19), (3.20), and (3.21) as KAPA-1, KAPA-2, KAPA-3, and KAPA-4, respectively. Each will be treated independently in the sequel.

3.2.1 KAPA-1 (Simple KAPA)

Recursion equation (3.18) uses the stochastic gradient descent and is the simplest among all. It is, hence, also named simple KAPA. The same methodology for KLMS is used to rewrite (3.18) as a sum of errors multiplied by the transformed inputs. If we set the initial guess $\boldsymbol{\omega}(0) = 0$, then the iteration of equation (3.18) will be

$$\begin{aligned}
&\boldsymbol{\omega}(0) = 0\\
&\boldsymbol{\omega}(1) = \eta d(1)\boldsymbol{\varphi}(0) = \mathbf{a}_1(1)\boldsymbol{\varphi}(1)\\
&\cdots\\
&\boldsymbol{\omega}(i-1) = \sum_{j=1}^{i-1}\mathbf{a}_j(i-1)\boldsymbol{\varphi}(j)\\
&\boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1) = \left[\sum_{j=1}^{i-1}\mathbf{a}_j(i-1)\kappa_{i-K+1,j}, \ldots, \sum_{j=1}^{i-1}\mathbf{a}_j(i-1)\kappa_{i-1,j}, \sum_{j=1}^{i-1}\mathbf{a}_j(i-1)\kappa_{i,j}\right]^T\\
&\mathbf{e}(i) = \mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)\\
&\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)\mathbf{e}(i) = \sum_{j=1}^{i-1}\mathbf{a}_j(i-1)\boldsymbol{\varphi}(j) + \sum_{j=1}^{K}\eta\mathbf{e}_j(i)\boldsymbol{\varphi}(i+K-j)
\end{aligned} \quad (3.22)$$

where $k_{i,j} = k(\mathbf{u}(i), \mathbf{u}(j))$ for simplicity.

Note that during the iteration, the weight vector in the feature space assumes the following expansion:

$$\boldsymbol{\omega}(i) = \sum_{j=1}^{i} \mathbf{a}_j(i)\boldsymbol{\varphi}(j) \quad \forall i > 0 \tag{3.23}$$

i.e., the weight at time i is a linear combination of the previous transformed input. This result may seem simply a restatement of the representer theorem in Schölkopf et al. [2001]. However, it should be emphasized that this result does not rely on any explicit minimal norm constraint as required for the representer theorem. As we discussed in Chapter 2, the gradient search has an inherent regularization mechanism that guarantees the solution is in the data subspace under appropriate initialization. In general, the initialization $\boldsymbol{\omega}(0)$ can alternatively translate whatever a priori information is available as long as it can be expressed as a linear combination of transformed data to use the kernel trick, but the solution loses its minimum norm property. By equation (3.23), the weight vector updating is accomplished through the expansion coefficients

$$\mathbf{a}_k(i) = \begin{cases} \eta\left(d(i) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i,j}\right), & k = i \\ \mathbf{a}_k(i-1) + \eta\left(d(k) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}\right), & i-K+1 \le k \le i-1 \\ \mathbf{a}_k(i-1), & 1 \le k < i-K+1 \end{cases} \tag{3.24}$$

Let us introduce a simplified notation $e(i;k) = e_{K+k-i}(i) = d(k) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}$ indicating the prediction error on data $\{\mathbf{u}(k), d(k)\}$ using $\boldsymbol{\omega}(i-1)$. The interpretation of equation (3.24) becomes straightforward: allocate a new unit with coefficient $\eta e(i;i)$ and update the coefficients for the other $K-1$ most recent units by $\eta e(i;k)$ for $i-K+1 \le k \le i-1$.

If we denote f_i as the estimate of the input-output mapping at time i, we have the following sequential learning rule for KAPA-1:

$$f_i = f_{i-1} + \eta \sum_{j=i-K+1}^{i} e(i;j)\kappa(\mathbf{u}(j), \cdot) \tag{3.25}$$

The coefficients $\mathbf{a}(i)$ and the centers $\mathcal{C}(i)$ should be stored in the computer during training. The updates needed for KAPA-1 at time i are

$$\begin{aligned} &\mathbf{a}_i(i) = \eta e(i;i) \\ &\mathbf{a}_j(i) = \mathbf{a}_j(i-1) + \eta e(i;j), \quad j = i-K+1, \ldots, i-1 \\ &\mathbf{a}_j(i) = \mathbf{a}_j(i-1), \quad j = 1, \ldots, i-K \\ &\mathcal{C}(i) = \{\mathcal{C}(i-1), \mathbf{u}(i)\} \end{aligned} \tag{3.26}$$

The pseudocode for KAPA-1 is listed in Algorithm 4.

Algorithm 4. The kernel affine projection algorithm–type 1 (KAPA-1)

Initialization

step-size parameter η
$\mathbf{a}_1(1) = \eta d(1)$

Computation

while $\{\mathbf{u}(i), d(i)\}$ available **do**
  %allocate a new unit
  $\mathbf{a}_i(i-1) = 0$
  for $k = \max(1, i-K+1)$ to i **do**
    %evaluate outputs of the current network
    $y(i;k) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}$
    %computer errors
    $e(i;k) = d(k) - y(i;k)$
    %update the min(i, K) most recent units
    $\mathbf{a}_k(i) = \mathbf{a}_k(i-1) + \eta e(i;k)$
  end for
  if $i > K$ **then**
    %keep the remaining
    for $k = 1$ to $i-K$ **do**
      $\mathbf{a}_k(i) = \mathbf{a}_k(i-1)$
    end for
  end if
end while

At iteration i, given a test point input $\mathbf{u}_*$, the system output is computed as

$$f(\mathbf{u}_*) = \sum_{j=1}^{i} \mathbf{a}_j(i)\kappa(\mathbf{u}(j), \mathbf{u}_*)$$

3.2.2 KAPA-2 (Normalized KAPA)

Similarly, the smoothed Newton's recursion [equation (3.19)] can be factorized into the following steps:

$$\begin{aligned}
\boldsymbol{\omega}(i-1) &= \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\boldsymbol{\varphi}(j) \\
\mathbf{e}(i) &= \mathbf{d}(i) - \boldsymbol{\Phi}(i)^T \boldsymbol{\omega}(i-1) \\
\mathbf{G}(i) &= \boldsymbol{\Phi}(i)^T \boldsymbol{\Phi}(i) \\
\boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\mathbf{G}(i) + \varepsilon\mathbf{I}]^{-1}\mathbf{e}(i)
\end{aligned} \tag{3.27}$$

In practice, we do not have access to the transformed weight $\boldsymbol{\omega}$ or any transformed data, so the update has to be evaluated through the expansion

coefficient $\mathbf{a}$, just like in KAPA-1. The whole recursion is similar to equation (3.24) except that the error is normalized by a $K \times K$ matrix $[\mathbf{G}(i) + \varepsilon\mathbf{I}]^{-1}$.

3.2.3 KAPA-3 (Leaky KAPA)

When the cost function (3.17) is ill-posed in the conventional empirical risk minimization (ERM) sense [Girosi et al., 1995], the common practice is to constrain the solution norm

$$\min_{\boldsymbol{\omega}} \mathbf{E}\left|d - \boldsymbol{\omega}^T \boldsymbol{\varphi}(\mathbf{u})\right|^2 + \lambda \|\boldsymbol{\omega}\|^2 \tag{3.28}$$

As we have already shown in equation (3.20), the leaky KAPA is

$$\boldsymbol{\omega}(i) = (1 - \lambda\eta)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)\left[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T \boldsymbol{\omega}(i-1)\right] \tag{3.29}$$

Again, the iteration will be based on the expansion coefficient $\mathbf{a}$, which is similar to equation (3.24).

$$\mathbf{a}_k(i) = \begin{cases} \eta\left(d(i) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i,j}\right), & k = i \\ (1-\lambda\eta)\mathbf{a}_k(i-1) + \eta\left(d(k) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i,j}\right), & i-K+1 \le k \le i-1 \\ (1-\lambda\eta)\mathbf{a}_k(i-1), & 1 \le k < i-K+1 \end{cases} \tag{3.30}$$

The only difference with respect to KAPA-1 is that KAPA-3 has a scaling factor $(1 - \lambda\eta)$ multiplying the previous weight, which is less than 1, and it imposes a forgetting mechanism so that the training data in the far past are scaled down exponentially. Furthermore, because the network size is growing over training, a transformed data can be pruned from the expansion easily if its coefficient is smaller than some prespecified threshold.

3.2.4 KAPA-4 (Leaky KAPA with Newton's Recursion)

As before, KAPA-4 [equation (3.21)] reduces to

$$\mathbf{a}_k(i) = \begin{cases} \eta\tilde{d}(i), & k = i \\ (1-\eta)\mathbf{a}_k(i-1) + \eta\tilde{d}(k), & i-K+1 \le k \le i-1 \\ (1-\eta)\mathbf{a}_k(i-1), & 1 \le k < i-K+1 \end{cases} \tag{3.31}$$

where $\tilde{\mathbf{d}}(i) = (\mathbf{G}(i) + \lambda\mathbf{I})^{-1}\mathbf{d}(i)$.

Table 3.1. Comparison of KAPA update rules.

Algorithm	Update equation
KAPA-1	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\Phi(i)[\mathbf{d}(i) - \Phi(i)^T\boldsymbol{\omega}(i-1)]$
KAPA-2	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\Phi(i)[\Phi(i)^T\Phi(i) + \varepsilon\mathbf{I}]^{-1}[\mathbf{d}(i) - \Phi(i)^T\boldsymbol{\omega}(i-1)]$
KAPA-3	$\boldsymbol{\omega}(i) = (1-\lambda\eta)\boldsymbol{\omega}(i-1) + \eta\Phi(i)[\mathbf{d}(i) - \Phi(i)^T\boldsymbol{\omega}(i-1)]$
KAPA-4	$\boldsymbol{\omega}(i) = (1-\eta)\boldsymbol{\omega}(i-1) + \eta\Phi(i)[\Phi(i)^T\Phi(i) + \lambda\mathbf{I}]^{-1}\mathbf{d}(i)$

Among these four algorithms, the first three require the error information to update the network, which is computationally expensive; however, KAPA-4 does not. Therefore, the different update rule in KAPA-4 has a huge significance in terms of computation because it only needs a $K \times K$ matrix inversion, which by using the sliding-window trick, only requires $O(K^2)$ operations [Van Vaerenbergh et al. 2006].We summarize the four KAPA update equations in Table 3.1 for ease of comparison.

3.3 ERROR REUSING

As we observe in KAPA-1, KAPA-2, and KAPA-3, the most time-consuming part of the computation is to calculate the prediction errors. For example, suppose $\boldsymbol{\omega}(i-1) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\boldsymbol{\varphi}(j)$. We need to calculate

$$e(i;k) = d(k) - \boldsymbol{\omega}(i-1)^T\boldsymbol{\varphi}(k)$$

for $i - K + 1 \le k \le i$ to compute $\boldsymbol{\omega}(i)$, which consists of $(i-1)K$ kernel evaluations. As i increases, this dominates the computation time. In this sense, the computation complexity of KAPA is K times of KLMS. However, after careful manipulation, we can shrink the complexity gap between KAPA and KLMS by reusing the errors.

Assume that all the K errors

$$e(i-1;k) = d(k) - \boldsymbol{\omega}(i-2)^T\boldsymbol{\varphi}(k)$$

for $i - K \le k \le i - 1$ are stored from the previous iteration. At the present iteration, we have

$$\begin{aligned} e(i;k) &= d(k) - \boldsymbol{\varphi}(k)^T\boldsymbol{\omega}(i-1) \\ &= d(k) - \boldsymbol{\varphi}(k)^T\left[\boldsymbol{\omega}(i-2) + \eta\sum_{j=i-\boldsymbol{K}}^{i-1} e(i-1;j)\boldsymbol{\varphi}(j)\right] \\ &= \left[d(k) - \boldsymbol{\varphi}(k)^T\boldsymbol{\omega}(i-2)\right] + \eta\sum_{j=i-K}^{i-1} e(i-1;j)\kappa_{j,k} \\ &= e(i-1;k) + \sum_{j=i-K}^{i-1}\eta e(i-1;j)\kappa_{j,k} \end{aligned} \tag{3.32}$$

Note that $e(i-1;k)$, $k<i$ have all been computed previously. Therefore, the only term that is not available is $e(i-1;i)$, which requires $i-1$ kernel evaluations. Overall, the computation complexity of KAPA-1 is $O(i+K^2)$, which is only $O(K^2)$ more than KLMS.

3.4 SLIDING WINDOW GRAM MATRIX INVERSION

In KAPA-2 and KAPA-4, another computation difficulty is to invert a $K\times K$ matrix, which normally requires $O(K^3)$ operations. However, in KAPA, the data matrix $\Phi(i)$ has a sliding window structure; therefore, a trick can be used to speed up the computation. The trick is based on the matrix inversion formula and was introduced in Van Vaerenbergh et al. [2006].We outline the basic calculation steps here. Suppose the sliding matrices share the same submatrix $\mathbf{D}$

$$\mathbf{G}(i-1)+\lambda\mathbf{I}=\begin{bmatrix} a & \mathbf{b}^T \\ \mathbf{b} & \mathbf{D} \end{bmatrix},\quad \mathbf{G}(i)+\lambda\mathbf{I}=\begin{bmatrix} \mathbf{D} & \mathbf{h} \\ \mathbf{h}^T & g \end{bmatrix} \tag{3.33}$$

and we know from the previous iteration

$$(\mathbf{G}(i-1)+\lambda\mathbf{I})^{-1}=\begin{bmatrix} e & \mathbf{f}^T \\ \mathbf{f} & \mathbf{H} \end{bmatrix} \tag{3.34}$$

First, calculate the inverse of $\mathbf{D}$ as

$$\mathbf{D}^{-1}=\mathbf{H}-\mathbf{f}\mathbf{f}^T/e \tag{3.35}$$

Then, update the inverse of the new Gram matrix as

$$(\mathbf{G}(i)+\lambda\mathbf{I})^{-1}=\begin{bmatrix} \mathbf{D}^{-1}+(\mathbf{D}^{-1}\mathbf{h})(\mathbf{D}^{-1}\mathbf{h})^T s^{-1} & -(\mathbf{D}^{-1}\mathbf{h})s^{-1} \\ -(\mathbf{D}^{-1}\mathbf{h})^T s^{-1} & s^{-1} \end{bmatrix} \tag{3.36}$$

with $s=g-\mathbf{h}^T\mathbf{D}^{-1}\mathbf{h}$. The overall complexity is $O(K^2)$.

3.5 TAXONOMY FOR RELATED ALGORITHMS

3.5.1 KLMS Algorithm

If $K=1$, then KAPA-1 reduces to the following KLMS:

$$\boldsymbol{\omega}(i)=\boldsymbol{\omega}(i-1)+\eta\boldsymbol{\varphi}(i)\left[d(i)-\boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1)\right]$$

It is observed that the KLMS allocates a new unit when a new training data comes in, with the input $\mathbf{u}(i)$ as the center and the prediction error as the coefficient (scaled by the step-size parameter). In other words, once the unit is allocated, the coefficient is fixed. It mimics the resource-allocating step in the resource allocating network (RAN) algorithm, whereas it neglects the adaptation step. In this sense, the KAPA algorithms that allocate a new unit for the present input and adapt the other $K - 1$ most recent allocated units are closer to the original RAN.

Similarly, the normalized KLMS algorithm is a special case of KAPA-2 with $K = 1$:

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \frac{\eta \boldsymbol{\varphi}(i)}{\varepsilon + \kappa(\mathbf{u}(i), \mathbf{u}(i))}\left[d(i) - \boldsymbol{\varphi}(i)^T \boldsymbol{\omega}(i-1)\right] \tag{3.37}$$

Notice that for translation invariant kernels, i.e., $k(\mathbf{u}(i), \mathbf{u}(i)) = \text{const}$, KLMS is normalized automatically. Sometimes we use KLMS-1 and KLMS-2 to distinguish the two.

3.5.2 NORMA Algorithm

Similarly, KAPA-3 [equation (3.20)] reduces to the NORMA algorithm introduced by Kivinen et al. [2004].

$$\boldsymbol{\omega}(i) = (1 - \eta\lambda)\boldsymbol{\omega}(i-1) + \eta \boldsymbol{\varphi}(i)\left[d(i) - \boldsymbol{\varphi}(i)^T \boldsymbol{\omega}(i-1)\right] \tag{3.38}$$

As we discussed in Chapter 2, penalizing explicitly the solution norm introduces a bias and significantly degenerates the overall performance, so in general we do not recommend the use of KAPA-3.

3.5.3 Kernel ADALINE

Assume that the size of the training data is finite N. If we set $K = N$, then the update rule of KAPA-1 becomes

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta \boldsymbol{\Phi}\left[\mathbf{d} - \boldsymbol{\Phi}^T \boldsymbol{\omega}(i-1)\right]$$

where the full data matrices are

$$\boldsymbol{\Phi} = [\boldsymbol{\varphi}(1), \ldots, \boldsymbol{\varphi}(N)], \quad \mathbf{d} = [d(1), \ldots, d(N)]$$

It is easy to check that the weight vector also assumes the following expansion:

$$\boldsymbol{\omega}(i) = \sum_{j=1}^{N} \mathbf{a}_j(i) \boldsymbol{\varphi}(j)$$

Table 3.2. List of algorithms related to KAPA.

Algorithm	Update equation	Relation to KAPA
KLMS	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\varphi(i)[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-1, $K = 1$
NKLMS	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \dfrac{\eta\boldsymbol{\varphi}(i)}{(\varepsilon + \kappa_{i,i})}\left[d(i) - \boldsymbol{\varphi}(i)^T\boldsymbol{\omega}(i-1)\right]$	KAPA-2, $K = 1$
NORMA	$\boldsymbol{\omega}(i) = (1 - \eta\lambda)\boldsymbol{\omega}(i-1) + \eta\varphi(i)[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-3, $K = 1$
KA	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\Phi[\mathbf{d} - \Phi^T\boldsymbol{\omega}(i-1)]$	KAPA-1, $K = N$
SW-KRLS	$\boldsymbol{\omega}(i) = \Phi(i)[\Phi(i)^T\Phi(i) + \lambda\mathbf{I}]^{-1}\mathbf{d}(i)$	KAPA-4, $\eta = 1$
RegNet	$\boldsymbol{\omega}(i) = \Phi[\Phi^T\Phi + \lambda\mathbf{I}]^{-1}\mathbf{d}$	KAPA-4, $\eta = 1$, $K = N$

And the updating on the expansion coefficients is

$$\mathbf{a}_j(i) = \mathbf{a}_j(i-1) + \eta\left[d(j) - \boldsymbol{\varphi}(j)^T\boldsymbol{\omega}(i-1)\right]$$

This is nothing but the kernel ADALINE (KA) introduced by Frieb and Harrison [1999]. Notice that the kernel ADALINE is not an online method.

3.5.4 Sliding Window Kernel Recursive Least Squares

In KAPA-4, if we set $\eta = 1$, we have

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)\left[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \lambda\mathbf{I}\right]^{-1}\mathbf{d}(i) \tag{3.39}$$

which is the sliding-window kernel RLS (SW-KRLS) introduced by Van Vaerenbergh et al. [2006].

3.5.5 Regularization Networks

We assume there are only N training data and $K = N$. Equation (3.21) becomes directly

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}\left[\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}\right]^{-1}\mathbf{d} \tag{3.40}$$

which is the regularization network (RegNet) [Girosi et al., 1995].

We summarize all the related algorithms in Table 3.2 for convenience.

3.6 COMPUTER EXPERIMENTS

3.6.1 KAPA Applied to Mackey–Glass Time-Series Prediction

This example is another study on the short-term prediction of the Mackey–Glass (MG) chaotic time series discussed in Chapter 2. We set the time embed-

ding as 7 here; i.e., $\mathbf{u}(i) = [x(i-7), x(i-6), \ldots, x(i-1)]^T$ are used as the input to predict the present one $x(i)$.

Part 1: A segment of 500 samples is used as the training data and another 100 points as the test data (in the testing phase, the filter is fixed). All the data are corrupted by Gaussian noise with zero mean and 0.001 variance.

We compare the prediction performance of KLMS, KAPA-1, KAPA-2, KRLS, and a linear combiner trained with LMS. KRLS will be fully discussed in the next chapter of the book, and it is only presented here for comparison. The Gaussian kernel [equation (1.24)] with kernel parameter $a = 1$ is chosen for all the kernel-based algorithms. Figure 3.1 is a typical plot of the learning curves for the LMS, KLMS-1, KAPA-1, KAPA-2 ($K = 10$), and KRLS, respectively. The last 100 points of the learning curves are used to compute the results listed in Table 3.3, where the parameters of each algorithm are also listed.

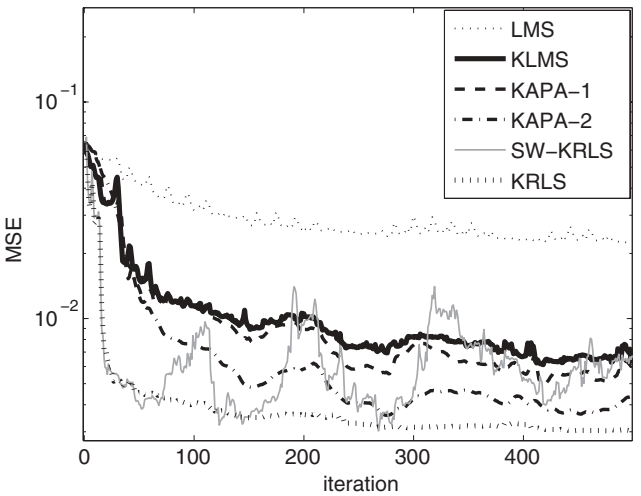


Figure 3.1. Learning curves of LMS, KLMS, KAPA-1 ($K = 10$), KAPA-2 ($K = 10$), SW-KRLS ($K = 50$), and KRLS in Mackey–Glass time-series prediction.

Table 3.3. Performance comparison of LMS, KLMS, KAPA, SW-KRLS, and KRLS in Mackey–Glass time-series prediction.

Algorithm	Parameters	Test mean square error
LMS	$\eta = 0.04$	0.0208 ± 0.0009
KLMS	$\eta = 0.02$	0.0052 ± 0.00022
SW-KRLS	$K = 50, \lambda = 0.1$	0.0052 ± 0.00026
KAPA-1	$\eta = 0.03, K = 10$	0.0048 ± 0.00023
KAPA-2	$\eta = 0.03, K = 10, \varepsilon = 0.1$	0.0040 ± 0.00028
KRLS	$\lambda = 0.1$	0.0027 ± 0.00009

Table 3.4. Complexity comparison of LMS, KLMS, KAPA, SW-KRLS, and KRLS at iteration *i*.

Algorithm	Computation	Memory
LMS	$O(L)$	$O(L)$
KLMS	$O(i)$	$O(i)$
SW-KRLS	$O(K^2)$	$O(K^2)$
KAPA-1	$O(i + K^2)$	$O(i + K)$
KAPA-2	$O(i + K^2)$	$O(i + K^2)$
KAPA-4	$O(K^2)$	$O(i + K^2)$
KRLS	$O(i^2)$	$O(i^2)$

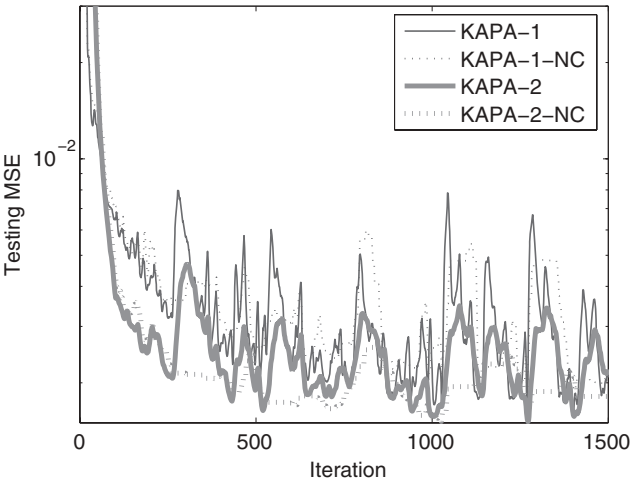


Figure 3.2. Learning curves of KAPA-1 (K = 10) and KAPA-2 (K = 10) with and without novelty criterion in Mackey–Glass time-series prediction.

As we can observe in Table 3.3, the performance of KAPA-2 is substantially better than KLMS. All the results in the tables are in the form of "average ± standard deviation." Table 3.4 summarizes the computational complexity of these algorithms. KLMS and KAPA effectively reduce the computational complexity and memory storage when compared with KRLS. KAPA-3 and SW-KRLS are also tested on this problem. It is observed that the performance of KAPA-3 is similar to KAPA-1 when the forgetting term is close to 1 as expected and the results are severely biased when the forgetting term is reduced even more. The performance of SW-KRLS is included in Figure 3.1 and in Table 3.3 with $K = 50$. It is observed that KAPA-4 (including SW-KRLS) does not perform well with small K (<50).

Part 2: We test how the novelty criterion affects the performance of KAPA algorithms. A segment of 1500 samples is used as the training data and another 200 as the test data. All the data are corrupted by Gaussian noise with zero mean and 0.0001 variance. The thresholds in the novelty criterion are set as $\delta_1 = 0.1$ and $\delta_2 = 0.05$. The learning curves are shown in Figure 3.2, and the

Table 3.5. Performance of KAPA with novelty criterion in Mackey–Glass time-series prediction.

Algorithm	Parameters	Test mean square error	Dictionary size
KAPA-1	$\eta = 0.05$	0.0026 ± 0.00069	1000
KAPA-1-NC	$\eta = 0.05$	0.0020 ± 0.00004	395
KAPA-2	$\eta = 0.05$, $\varepsilon = 0.1$	0.0022 ± 0.00041	1000
KAPA-2-NC	$\eta = 0.05$, $\varepsilon = 0.1$	0.0018 ± 0.00007	336

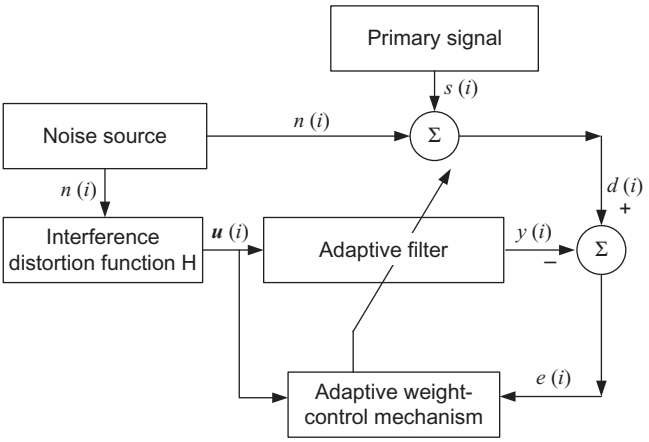


Figure 3.3. Basic structure of a noise cancellation system.

results are summarized in Table 3.5, which is calculated from the last 100 points of the learning curves. It is observed that the complexity can be reduced dramatically with the novelty criterion preserving the prediction accuracy. Here we use "-NC" to indicate the corresponding algorithms equipped with the novelty criterion.

3.6.2 KAPA Applied to Noise Cancellation

Another important problem in signal processing is noise cancellation in which an unknown interference has to be removed based on some reference measurement.[3] The basic structure of a noise cancellation system is shown in Figure 3.3. The primary signal is $s(i)$, and its noisy measurement $d(i)$ acts as the desired signal of the system. $n(i)$ is a white noise process that is unknown, and $u(i)$ is its reference measurement, i.e., a distorted version of the noise process through some distortion function, which is unknown in general. Here $u(i)$ is the input of the adaptive filter. The objective is to use $u(i)$ as the input to the filter and to obtain as the filter output an estimate of the noise source $n(i)$. Therefore, the noise can be subtracted from $d(i)$ to improve the signal-to-noise ratio (SNR).

Part 1: In this example, the noise source is assumed white, uniformly distributed between [–0.5, 0.5]. The interference distortion function is assumed to be

$$u(i) = n(i) - 0.2u(i-1) - u(i-1)n(i-1) + 0.1n(i-1) + 0.4u(i-2) \tag{3.41}$$

As we observe, the distortion function is nonlinear (multiplicative) and has infinite impulsive response, which means it is impossible to recover $n(i)$ from a finite time delay embedding of $u(i)$.We rewrite the distortion function as

$$n(i) = u(i) + 0.2u(i-1) - 0.4u(i-2) + (u(i-1) - 0.1)n(i-1)$$

Therefore, the current value of the noise source $n(i)$ not only depends on the reference noise measure $[u(i), u(i-1), u(i-2)]$ but also on the previous value $n(i-1)$, which in turn depends on $[u(i-1), u(i-2), u(i-3)]$, and so on. It means we need a long time embedding (infinitely long theoretically) to recover $n(i)$ accurately. However, the recursive nature of an adaptive system provides a feasible alternative; i.e., we feedback the output of the filter $\hat{n}(i-1)$, which is the estimate of $n(i-1)$ to estimate the present one, pretending $\hat{n}(i-1)$ is the true value of $n(i-1)$. Therefore, the input of the adaptive filter is of the form $[u(i), u(i-1), u(i-2), \hat{n}(i-1)]$. It can be observed that the system is inherently recurrent. In the linear case, it belongs to the *output error methods* [Goodwin and Sin, 1984]. However, it will be nontrivial to generalize the results concerning convergence and stability to nonlinear cases, and it serves as a line of future work.

We assume the primary signal $s(i) = 0$ during the training phase. And the system simply tries to reconstruct the noise source from the reference measurement. We use a linear filter trained with normalized LMS (NLMS), and two nonlinear filters trained with KLMS-NC and KAPA-2-NC ($K = 10$), respectively. One thousand training samples are used and 200 Monte Carlo simulations are run to get the ensemble learning curves as shown in Figure 3.4. The step-size parameter and regularization parameter for NLMS is 0.2 and 0.005, respectively. The step-size parameter for KLMS-NC and KAPA-2-NC is 0.5 and 0.2, respectively. The Gaussian kernel is used for both KLMS-NC and KAPA-2-NC with kernel parameter $a = 1$. The tolerance parameters for KLMS-NC and KAPA-2-NC are $\delta_1 = 0.15$ and $\delta_2 = 0.01$. The noise reduction (NR) factor, which is defined as $10 \log_{10}\{E[n^2(i)]/E[n(i) - y(i)]^2\}$, is listed in Table 3.6 along with the corresponding network size (the final number of centers). The performance improvement of KAPA-2-NC is obvious when compared with KLMS-NC.

Part 2: Next, we use a more realistic noise source (instead of the white noise), which is a functional magnetic resonance imaging (fMRI) recording provided by Dr. Issa Panahi from University of Texas at Dallas. The mean of the fMRI noise is 0 and the standard deviation is 0.051. The typical waveform is shown in Figure 3.5. We compare KAPA-2-NC with NLMS. Two hundred Monte Carlo simulations are conducted using different segments of

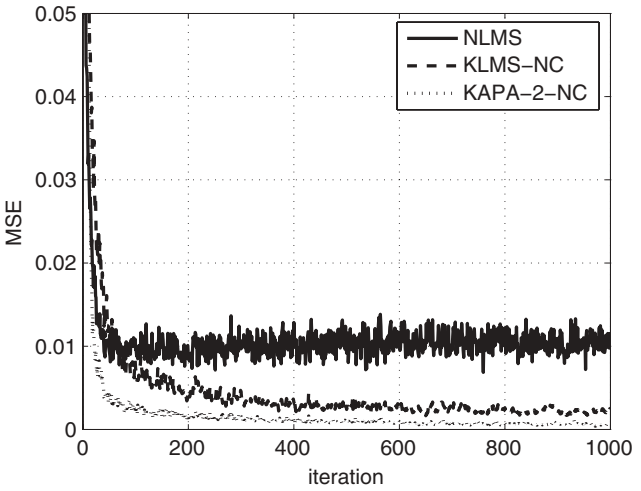


Figure 3.4. Ensemble learning curves of NLMS, KLMS-NC and KAPA-2-NC (K = 10) in noise cancellation.

Table 3.6. Performance comparison of NLMS, KLMS, and KAPA-2 in noise cancellation.

Algorithm	Network size	NR(dB)
NLMS	N/A	9.09 ± 0.45
KLMS-NC	407 ± 14	15.58 ± 0.48
KAPA-2-NC	370 ± 14	21.99 ± 0.80

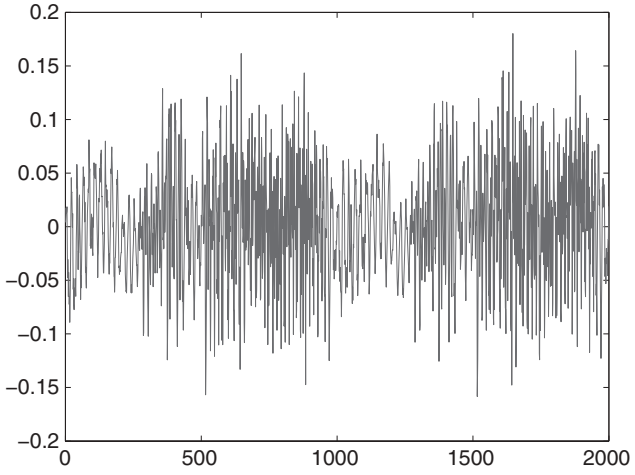


Figure 3.5. A typical segment of fMRI noise recording.

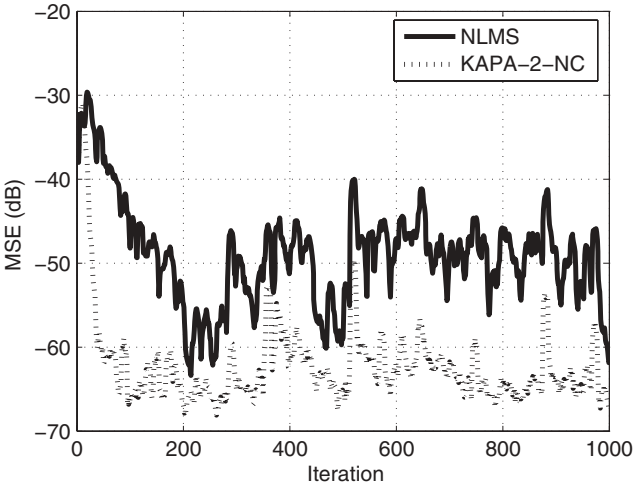


Figure 3.6. Ensemble learning curves of NLMS and KAPA-2-NC (K = 10) in fMRI noise cancellation.

Table 3.7. Performance comparison of NLMS and KAPA-2 using actual fMRI noise recording.

Algorithm	Network size	NR(dB)
NLMS	N/A	23.68 ± 4.14
KAPA-2-NC	170 ± 12	36.50 ± 2.29

the recording. We average all the learning curves together to get the ensemble learning curves plotted in Figure 3.6. The step-size parameter and regularization parameter for NLMS is 0.2 and 0.005, respectively. The step-size parameter for KAPA-2-NC is 0.2. The Gaussian kernel is used for KAPA-2-NC with kernel parameter $a = 1$. The tolerance parameters are $\delta_1 = 0$ and $\delta_2 = 0.001$. And the NR is listed in Table 3.7 along with the corresponding network size (the final number of units). The performance improvement of KAPA-2-NC is significant when compared with NLMS.

3.6.3 KAPA Applied to Nonlinear Channel Equalization

In this example, we reconsider the nonlinear channel equalization problem (see Figure 3.7). The problem setting is the same as in Chapter 2: A binary signal $\{s(1), s(2), \ldots, s(N)\}$ is fed into the nonlinear channel. At the receiver end of the channel, the signal is corrupted by additive white Gaussian noise and is then observed as $\{r(1), r(2), \ldots, r(N)\}$. The aim of channel equalization is to construct an *inverse* filter that reproduces the original signal with as low an error rate as possible. It is easy to formulate it as a regression problem, with input–output examples $\{(r(i + D), r(i + D - 1), \ldots, r(i + D - l + 1)), s(i)\}$,

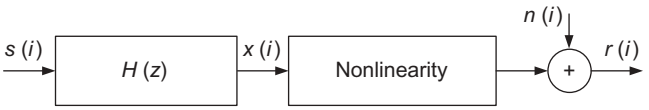


Figure 3.7. Basic structure of a nonlinear channel.

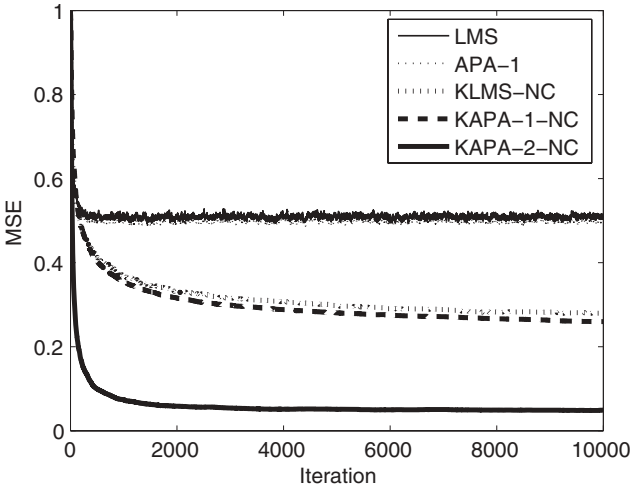


Figure 3.8. Ensemble learning curves of LMS, APA-1, KLMS-NC, KAPA-1-NC, and KAPA-2-NC in nonlinear channel equalization ($\sigma = 0.1$).

where l is the time embedding length and D is the equalization time lag. $l = 3$ and $D = 2$ in the equalizer.

Part 1: In this experiment, the nonlinear channel model is defined by $x(i) = s(i) + 0.5s(i-1)$, $r(i) = x(i) - 0.9x(i)^2 + n(i)$, where $n(i)$ is the white Gaussian noise with a variance of σ^2. We compare the performance of LMS, APA-1, KLMS-NC, KAPA-1-NC ($K = 10$), and KAPA-2-NC ($K = 10$). The Gaussian kernel with $a = 0.1$ is used in KLMS-NC, KAPA-1-NC, and KAPA-2-NC. The noise variance is fixed here at $\sigma = 0.1$. The ensemble learning curves are plotted in Figure 3.8 with 50 Monte Carlo simulations. For each Monte Carlo simulation, the learning curves are calculated on a segment of 100 testing data. The MSE is calculated between the continuous output (before taking the hard decision) and the desired signal. The performance of LMS and APA-1 is similar, and the two learning curves almost overlap. For KLMS-NC, KAPA-1-NC, and KAPA-2-NC, the novelty criterion is employed with $\delta_1 = 0.26$ and $\delta_2 = 0.08$. The dynamic change of the network size is also plotted in Figure 3.9 over the training. It can be observed that at the beginning, the network sizes increase quickly, but after convergence, the network sizes increase slowly. And in fact, we can stop adding new centers after convergence by noticing that the MSE does not change after convergence.

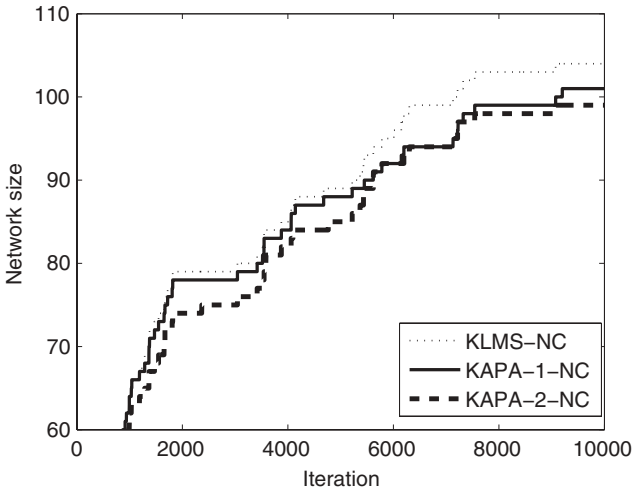


Figure 3.9. Network sizes of KLMS-NC, KAPA-1-NC, and KAPA-2-NC over training in nonlinear channel equalization.

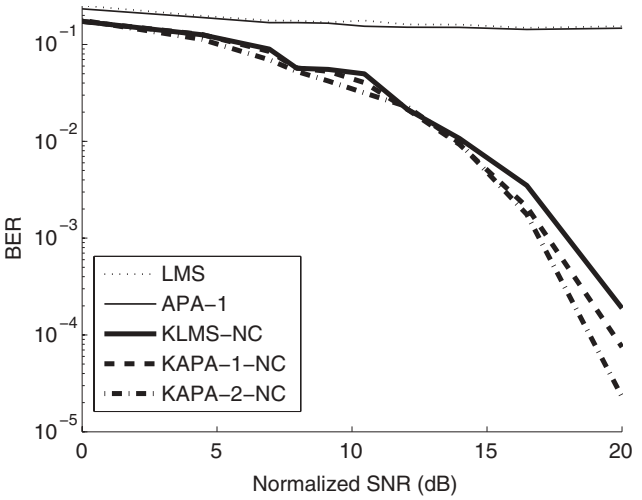


Figure 3.10. Performance comparison of LMS, APA-1, KLMS-NC, KAPA-1-NC, and KAPA-2-NC with different SNR in nonlinear channel equalization.

Part 2: Next, different noise variances are set. To make the comparison fair, we tune the novelty criterion parameters to make the network size almost the same (around 100) in each scenario by cross-validation. For each setting, 20 Monte Carlo simulations are run with different training data and different testing data. The size of the training data is 1000 and the size of the testing data is 10^5. The filters are fixed during the testing phase. The results are presented in Figure 3.10. The normalized SNR is defined as $10\log_{10}\frac{1}{\sigma^2}$. It

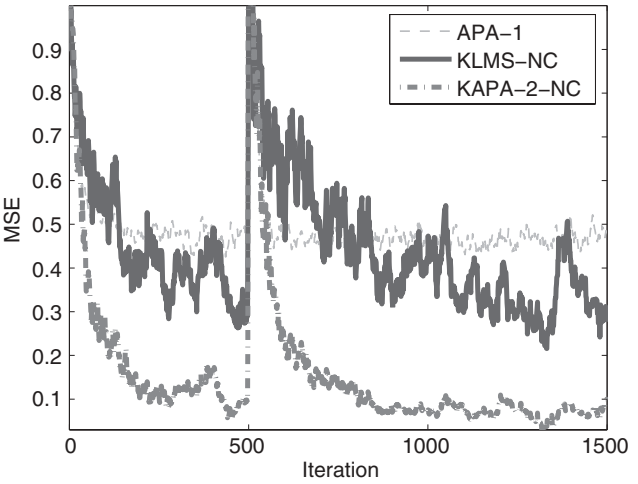


Figure 3.11. Ensemble learning curves of APA-1, KLMS-NC, and KAPA-2-NC with an abrupt change at iteration 500 in nonlinear channel equalization.

is shown clearly that KAPA-2-NC outperforms the KLMS-NC substantially in terms of bit error rate (BER). The linear methods never really work in this simulation regardless of SNR. The improvement of KAPA-1-NC on KLMS-NC is marginal, but it exhibits a smaller variance. The roughness in the curves is mostly caused by the variance from the stochastic training.

Part 3: In the last simulation, we test the tracking ability of the proposed methods by introducing an abrupt channel change during training. The size of training data is 1500. For the first 500 data, the channel model is kept the same as before, but for the last 1000 data, the nonlinearity of the channel is switched to $r(i) = -x(i) + 0.9x(i)^2 + n(i)$. The ensemble learning curves from 100 Monte Carlo simulations are plotted in Figure 3.11, and the dynamic change of the network size is plotted in Figure 3.12. It is observed that the KAPA-2-NC outperforms other methods with its fast tracking speed. KAPA-1-NC and KLMS-NC perform similarly in this example. It is also noted that the network sizes increase right after the change to the channel model.

3.7 CONCLUSION

This chapter discusses the KAPA algorithm family, which is a stochastic gradient methodology to solve least-squares problems in RKHS. Because the KAPA update equations can be written as inner products, KAPA can be computed efficiently in the input space. Similar algorithms are discussed in Richard et al. [2009] and Slavakis and Theodoridis [2008] from different perspectives.[4]

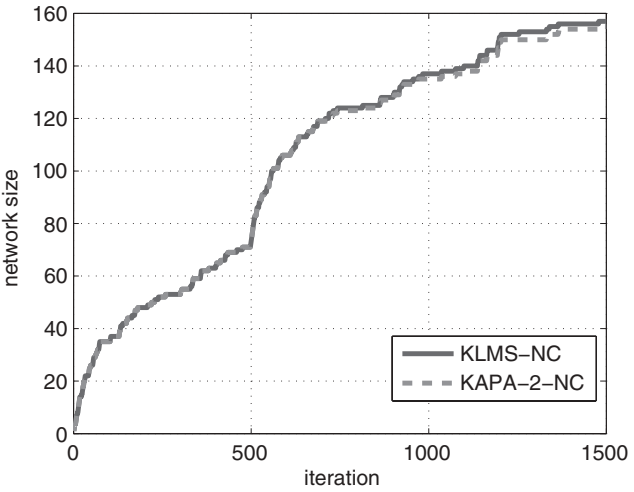


Figure 3.12. Network sizes of KLMS-NC and KAPA-2-NC over training with an abrupt change at iteration 500 in nonlinear channel equalization.

Compared with the simplest online gradient decent algorithm in RKHS (KLMS), and perhaps the most complex (KRLS), the KAPA family provides a flexible way of calculating a nonlinear filter online where the user can choose the performance/complexity trade-off at the point required by its application. Performance-wise KAPA is somewhere between KLMS and KRLS, which can be controlled by the window length K. The window length also controls the computational complexity. The relative performance was practically demonstrated in several important adaptive filtering applications, namely, time-series prediction, nonlinear channel equalization and nonlinear noise cancellation.

Moreover, the KAPA family also provides another theoretical understanding of radial basis function (RBF) like neural networks, including the batch regularized network, and it establishes relationships with a wealth of other algorithms available in the literature. Therefore, its role in building the taxonomy is also important and was well demonstrated in this chapter. We also illustrated the result of choosing the samples to keep in the filter using the novelty criterion. This simple criterion provides a large decrease in the number of samples with only a minor penalty in performance for the appropriate values of the thresholds, which are data dependent. This means that there is hope of decreasing even more the computational complexity with more sophisticated criteria.

Our emphasis in online adaptation is not accidental. Online algorithms are a necessity in many engineering applications (system identification of time-varying systems, channel tracking and equalization, echo cancellation, etc). However, we submit that online algorithms will also become increasingly more

useful for batch machine learning algorithms applied to large databases. In fact, the database sizes will continue to grow exponentially. This poses real problems for the algorithm designer because of the higher than linear increase in memory and computational complexity of batch algorithms [$O(N^2)$ and even $O(N^3)$ in some kernel algorithms]. Today, we still can afford these algorithmic complexities because the data sizes are reasonably small, but soon this will be unbearable because of the exponential growth of database sizes. Designers will be limited to sub $O(N^2)$ computation complexities [i.e., $O(N)$, $O(NlogN)$], which requires a paradigm shift in algorithm design. Online algorithms in kernel spaces will be in the critical path to sample these large databases stochastically, and find solutions quickly in the neighborhood of the optimum.

We will briefly make here the point more clearly. Learning theory is driven by two major theories: statistical learning and optimization. Each contributes to errors in the optimal solution. In fact, although one would like to minimize the expected risk, we settle by minimizing the empirical risk incurring an error (the estimation error) on the way. Moreover, the function that minimizes the empirical risk may not belong to the class of functions covered by the system, and we incur again an error (the approximation error). However, finding the optimal parameters of this system may be too complex to handle because of memory constraints or huge computational complexity. What we, among others [Bottou, 2008], are advocating is to incur a third error (the computation error) that finds parameters in the neighborhood of the optimum efficiently. The KAPA algorithms are exactly the enablers of this last step. Although we are still far from linear complexity algorithms, this seems to be a productive and relevant research direction.

ENDNOTES

1. **Affine Projection Algorithm.** The affine projection algorithm, from Ozeki and Umeda [1984], is a generalization and improvement of the well-known normalized least mean square algorithm. Following this early work, Gay and Tavathia [1995] described a fast implementation of the affine projection algorithm in the time domain, which features LMS-like complexity and RLS-like convergence in speech signal processing. Tanaka et al. [1999] proposed another fast implementation of the algorithm called *block exact fast affine projection*, using the frequency-domain approach; the algorithm exploits a fast finite impulse response (FIR) filtering technique based on the idea of fast convolution that uses the fast Fourier transform algorithm. Sankaran and Beex [2000] presented an analysis of convergence behavior of the algorithm with the following conclusions:

   - The learning curve of an affine projection adaptive filter consists of the sum of exponential terms.
   - An affine projection adaptive filter converges at a rate faster than that of the corresponding normalized LMS filter.

- As more delayed inputs are used, the rate of convergence improves, but the rate at which improvement is attained decreases.

For a discussion of regularization in fast affine projection implementation, please see Rombouts and Moonen [2000].

2. **Affine Projection Interpretation.** For our own study reported in the chapter, we use the recursion of APA-2 to explain why it can be interpreted as a projection problem onto an affine space. We roughly follow the derivation in Sayed [2003]. Define two estimation error vectors: the a priori output estimation error

$$\mathbf{e}(i) = \mathbf{d}(i) - \mathbf{U}(i)^T \mathbf{w}(i-1) \tag{3.42}$$

and the a posteriori output estimation error

$$\mathbf{r}(i) = \mathbf{d}(i) - \mathbf{U}(i)^T \mathbf{w}(i) \tag{3.43}$$

Then, it can be shown that the recursion of APA-2 [equation (3.10)] is the exact solution to the following *local* optimization problem:

$$\begin{aligned} &\min_{\mathbf{w}(i)} \|\mathbf{w}(i) - \mathbf{w}(i-1)\|^2 \\ &\text{subject to} \quad \mathbf{r}(i) = \left(\mathbf{I} - \eta \mathbf{U}(i)^T \mathbf{U}(i)\left[\mathbf{U}(i)^T \mathbf{U}(i) + \varepsilon \mathbf{I}\right]^{-1}\right)\mathbf{e}(i) \end{aligned} \tag{3.44}$$

In other words, we seek a $\mathbf{w}(i)$ that is closest to $\mathbf{w}(i-1)$ in the Euclidean norm sense and subject to an equality constraint between $\mathbf{r}(i)$ and $\mathbf{e}(i)$. This constraint guarantees that $\mathbf{U}(i)^T\mathbf{w}(i)$ will be a better estimate for $\mathbf{d}(i)$ than $\mathbf{U}(i)^T\mathbf{w}(i-1)$ for any step-size parameter η in the interval $(0, 2)$.

A special case of the formulation in equation (3.44) admits an interpretation in terms of projections onto affine subspaces. Setting $\eta = 1$ and $\varepsilon = 0$ in equation (3.44) yields

$$\min_{\mathbf{w}(i)} \|\mathbf{w}(i) - \mathbf{w}(i-1)\|^2 \quad \text{subject to } \mathbf{r}(i) = \mathbf{0} \tag{3.45}$$

or equivalently,

$$\min_{\mathbf{w}(i)} \|\mathbf{w}(i) - \mathbf{w}(i-1)\|^2 \quad \text{subject to } \mathbf{d}(i) = \mathbf{U}(i)^T \mathbf{w}(i) \tag{3.46}$$

A geometric interpretation of this equation is as follows: For any given data set $\{\mathbf{d}(i), \mathbf{U}(i)\}$, there may be infinitely many vector $\mathbf{w}$ that solve $d(i) = \mathbf{U}(i)^T\mathbf{w}$. The set of all such $\mathbf{w}$ is an affine subspace, or more precisely the intersection of K affine subspaces (it does not necessarily pass through the origin $\mathbf{w} = \mathbf{0}$). Given $\mathbf{w}(i-1)$, APA-2 selects that particular vector $\mathbf{w}(i)$ from this subspace that is closest to $\mathbf{w}(i-1)$ in the Euclidean norm sense. We, therefore, say that $\mathbf{w}(i)$ is obtained as the projection of $\mathbf{w}(i-1)$ onto the affine subspace.

3. **Adaptive Noise Cancellation.** Adaptive echo canceler and the adaptive line enhancer may be viewed as examples of the adaptive noise canceler, although they may be

intended for different applications. The initial work on adaptive echo cancelers started around 1965. Sondhi [1967] recognized that Kelly of Bell Telephone Laboratories was the first to propose the use of an adaptive filter for echo cancelation. This invention and its refinement are described in patents by Kelly and Logan [1970] and Sondhi [1970]. The adaptive line enhancer was originated by Widrow and his coworkers at Stanford University. Widrow et al. [1975] reported their early work of building a device to cancel 60-Hz interference at the output of an electrocardiographic amplifier and recorder in 1965. The adaptive line enhancer and its application as an adaptive detector were patented by McCool et al. [1980].

The first adaptive nonlinear noise cancelation appeared in Coker and Simkins [1980], where a simple nonlinear extension of the tapped delay line filter was trained by the LMS algorithm. Stapleton and Bass [1985] investigated a simple cascade model of a memoryless nonlinearity and a linear filter in the application of nonlinear noise control. More recently, recurrent radial-basis function networks [Billings and Fung, 1995], Volterra series [Li and Jiang, 2001], and fuzzy neural networks [Er et al., 2005] have also been investigated for adaptive noise cancellation.

4. **Kernel Affine Projection Algorithms.** Slavakis and Theodoridis [2008] derived a generalization of kernel affine projection algorithm based on the adaptive projected subgradient method. Classification is performed by metric projection mappings and sparsification is achieved by orthogonal projections, whereas online memory requirements and tracking are attained by oblique projections. The resulting sparsification scheme is similar to the classic sliding window adaptive schemes.

   Richard et al. [2009] presented a similar algorithm using the idea of local optimization in equation (3.46). A sparsification method called coherence criterion was discussed to control the size of the network. The coherence criterion is similar to the novelty criterion.

4

KERNEL RECURSIVE LEAST-SQUARES ALGORITHM

In this chapter, the kernel recursive least-squares algorithm (KRLS) will be discussed in detail, along with another sparsification approach called *approximate linear dependency*. The derivation is based on a least-squares formulation in the feature space as in the previous chapters.

We begin by revisiting the recursive least-squares (RLS) algorithm and map the RLS recursion into the feature space by using kernel mapping. Then, by exploiting a relation in matrix algebra known as the matrix inversion lemma, we develop the KRLS algorithm. An important feature of this algorithm is that its rate of convergence is typically an order of magnitude faster than that of the simple kernel least-mean-square algorithm (KLMS). This improvement in performance, however, is achieved at the expense of an increase in computational complexity.

In the second half of the chapter, we present a closely related methodology called Gaussian process regression (GPR). Although GPR and KRLS are mathematically equivalent, GPR has a distinct advantage of being principled and probabilistic. With the Bayesian interpretation, we present a well-founded framework for kernel design called maximum marginal likelihood (MML).

4.1 RECURSIVE LEAST-SQUARES ALGORITHM

With a sequence of training data $\{\mathbf{u}(j), d(j)\}_{j=1}^{i-1}$ up to and including time $i-1$, the recursive least-squares algorithm estimates the weight $\mathbf{w}(i-1)$ by minimizing a cost function as follows:

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

$$\min_{\mathbf{w}} \sum_{j=1}^{i-1} \left| d(j) - \mathbf{u}(j)^T \mathbf{w} \right|^2 \tag{4.1}$$

Here, $\mathbf{u}(j)$ is the $L \times 1$ regressor input and $d(j)$ is the desired response. Denote

$$\mathbf{U}(i-1) = [\mathbf{u}(1), \ldots, \mathbf{u}(i-1)]_{L \times (i-1)} \quad \text{and} \quad \mathbf{d}(i-1) = [d(1), \ldots, d(i-1)]^T$$

The solution to equation (4.1) is given by

$$\mathbf{w}(i-1) = \left(\mathbf{U}(i-1)\mathbf{U}(i-1)^T \right)^{-1} \mathbf{U}(i-1)\mathbf{d}(i-1) \tag{4.2}$$

When a new input–output pair $\{\mathbf{u}(i), d(i)\}$ becomes available, the weight estimate $\mathbf{w}(i)$, which is the minimizer of

$$\min_{\mathbf{w}} \sum_{j=1}^{j} \left| d(j) - \mathbf{u}(j)^T \mathbf{w} \right|^2 \tag{4.3}$$

is computed as

$$\mathbf{w}(i) = \left(\mathbf{U}(i)\mathbf{U}(i)^T \right)^{-1} \mathbf{U}(i)\mathbf{d}(i).$$

where

$$\mathbf{U}(i) = [\mathbf{U}(i-1), \mathbf{u}(i)] \quad \text{and} \quad \mathbf{d}(i) = \left[\mathbf{d}(i-1)^T, d(i) \right]^T$$

Define the matrices

$$\mathbf{P}(i) = \left(\mathbf{U}(i)\mathbf{U}(i)^T \right)^{-1} \quad \text{and} \quad \mathbf{P}(i-1) = \left(\mathbf{U}(i-1)\mathbf{U}(i-1)^T \right)^{-1}$$

for which we observe that

$$\mathbf{P}(i)^{-1} = \mathbf{P}(i-1)^{-1} + \mathbf{u}(i)\mathbf{u}(i)^T$$

Then, by using the matrix inversion lemma

$$(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{C}^{-1} + \mathbf{DA}^{-1}\mathbf{B} \right)^{-1}\mathbf{DA}^{-1} \tag{4.4}$$

with the identifications

$$\mathbf{P}(i-1) \to \mathbf{A}, \mathbf{u}(i) \to \mathbf{B}, 1 \to \mathbf{C}, u(i)^T \to \mathbf{D}$$

we have the following recursion to update $\mathbf{P}(i)$ directly from $\mathbf{P}(i-1)$:

$$\mathbf{P}(i) = \left[\mathbf{P}(i-1) - \frac{\mathbf{P}(i-1)\mathbf{u}(i)\mathbf{u}(i)^T\mathbf{P}(i-1)}{1 + \mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)} \right] \tag{4.5}$$

Furthermore, we obtain the recursion to update $\mathbf{w}(i)$ directly from the previous estimate $\mathbf{w}(i-1)$, as shown by

$$\begin{aligned}
\mathbf{w}(i) &= \mathbf{P}(i)\mathbf{U}(i)\mathbf{d}(i) \\
&= \left[\mathbf{P}(i-1) - \frac{\mathbf{P}(i-1)\mathbf{u}(i)\mathbf{u}(i)^T\mathbf{P}(i-1)}{1+\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)}\right][\mathbf{U}(i-1)\mathbf{d}(i-1) + \mathbf{u}(i)d(i)] \\
&= \underbrace{\mathbf{P}(i-1)\mathbf{U}(i-1)\mathbf{d}(i-1)}_{\mathbf{w}(i-1)} - \frac{\mathbf{P}(i-1)\mathbf{u}(i)\mathbf{u}(i)^T}{1+\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)}\underbrace{\mathbf{P}(i-1)\mathbf{U}(i-1)\mathbf{d}(i-1)}_{\mathbf{w}(i-1)} \\
&\quad + \mathbf{P}(i-1)\mathbf{u}(i)d(i) - \frac{\mathbf{P}(i-1)\mathbf{u}(i)\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)d(i)}{1+\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)} \\
&= \mathbf{w}(i-1) - \frac{\mathbf{P}(i-1)\mathbf{u}(i)\mathbf{u}(i)^T\mathbf{w}(i-1)}{1+\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)} \\
&\quad + \left[\mathbf{P}(i-1)\mathbf{u}(i)d(i) - \frac{\mathbf{P}(i-1)\mathbf{u}(i)\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)d(i)}{1+\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)}\right] \\
&= \mathbf{w}(i-1) - \frac{\mathbf{P}(i-1)\mathbf{u}(i)\mathbf{u}(i)^T\mathbf{w}(i-1)}{1+\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)} + \frac{\mathbf{P}(i-1)\mathbf{u}(i)d(i)}{1+\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)}
\end{aligned}$$

That is

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \frac{\mathbf{P}(i-1)\mathbf{u}(i)}{1+\mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i)}\left[d(i) - \mathbf{u}(i)^T\mathbf{w}(i-1)\right] \tag{4.6}$$

Define

$$r(i) = 1 + \mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i) \tag{4.7}$$

$$\mathbf{k}(i) = \mathbf{P}(i-1)\mathbf{u}(i)/r(i) \tag{4.8}$$

Then the standard RLS is summarized in Algorithm 5.

Algorithm 5. The recursive least-squares algorithm

Initialization

$\mathbf{w}(0) = 0$, $\mathbf{P}(0)$

Computation

Iterate for $i \geq 1$

$$\begin{aligned}
r(i) &= 1 + \mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i) \\
\mathbf{k}(i) &= \mathbf{P}(i-1)\mathbf{u}(i)/r(i) \\
e(i) &= d(i) - \mathbf{u}(i)^T\mathbf{w}(i-1) \\
\mathbf{w}(i) &= \mathbf{w}(i-1) + \mathbf{k}(i)e(i) \\
\mathbf{P}(i) &= \left[\mathbf{P}(i-1) - \mathbf{k}(i)\mathbf{k}(i)^T r(i)\right]
\end{aligned} \tag{4.9}$$

$\mathbf{k}(i)$ is called the gain vector, and $e(i)$ is the prediction error. Notice that $\mathbf{P}(i)$ is $L \times L$, where L is the dimensionality of the input $\mathbf{u}$. Therefore, the time and memory complexities are both $O(L^2)$. RLS distributes the computation load evenly into each iteration, which is appealing in applications like channel equalization where data are available sequentially over time.

4.1.1 Regularization and Initialization

Notice that in RLS, the matrix $\mathbf{P}(i)$ is defined as the inverse of the data autocorrelation matrix. However, this inverse may not exist especially during the initial update stages when $\mathbf{U}(i)$ has fewer columns than rows. A possible workaround is to collect a subset of data at the early stages of training and to estimate explicitly the inverse of the data autocorrelation matrix, which can be used as the initial value of $\mathbf{P}(i)$ in Algorithm 5. However, even with good initialization, the data autocorrelation matrix may become rank deficient at later stages. To overcome this difficulty, regularization is need, which leads to the following formulation:

$$\min_{\mathbf{w}} \sum_{j=1}^{i} \left| d(j) - \mathbf{u}(j)^T \mathbf{w} \right|^2 + \lambda \|\mathbf{w}\|^2 \tag{4.10}$$

where $\lambda \|\mathbf{w}\|^2$ is the *norm penalizing term* or the *regularization term*. λ is a positive number, called the *regularization parameter*. The solution to this problem is similarly expressed as

$$\mathbf{w}(i) = \left(\mathbf{U}(i)\mathbf{U}(i)^T + \lambda \mathbf{I} \right)^{-1} \mathbf{U}(i)\mathbf{d}(i) \tag{4.11}$$

where $\mathbf{I}$ the L-by-L identity matrix. And now

$$\mathbf{P}(i) = \left(\mathbf{U}(i)\mathbf{U}(i)^T + \lambda \mathbf{I} \right)^{-1} \tag{4.12}$$

which is guaranteed to be invertible. Therefore, the initial value can be simply set as

$$\mathbf{P}(0) = \lambda^{-1}\mathbf{I}.$$

Other equations remain unchanged in Algorithm 5.

In general, the regularization term can be in the form of $\mathbf{w}^T \Sigma_r \mathbf{w}$ with any positive-definite matrix Σ_r. Notice that the effect of regularization diminishes as more and more data are supplied.

4.2 EXPONENTIALLY WEIGHTED RECURSIVE LEAST-SQUARES ALGORITHM

In practice, an exponentially weighted mechanism in RLS is commonly used to put more emphasis on recent data and to deemphasize data from the remote past. Let β be a positive scalar, usually close to one:

$$0 \ll \beta \leq 1.$$

The weighted regularized least-squares cost function is now defined as

$$\min_{\mathbf{w}} \sum_{j=1}^{i} \beta^{i-j} \left| d(j) - \mathbf{u}(j)^T \mathbf{w} \right|^2 + \beta^i \lambda \|\mathbf{w}\|^2 \tag{4.13}$$

β is usually called the *forgetting factor*. Observe that the regularization term is weighted by β^i, which deemphasizes regularization as time progresses. Similarly, we may express the solution as

$$\mathbf{w}(i) = \left(\mathbf{U}(i) \mathbf{B}(i) \mathbf{U}(i)^T + \beta^i \lambda \mathbf{I} \right)^{-1} \mathbf{U}(i) \mathbf{B}(i) \mathbf{d}(i) \tag{4.14}$$

where

$$\mathbf{B}(i) = \text{diag}\left\{ \beta^{i-1}, \beta^{i-2}, \ldots, 1 \right\}$$

If we now define the quantities

$$\begin{aligned} \mathbf{P}(i) &= \left(\mathbf{U}(i) \mathbf{B}(i) \mathbf{U}(i)^T + \beta^i \lambda \mathbf{I} \right)^{-1} \\ r(i) &= 1 + \beta^{-1} \mathbf{u}(i)^T \mathbf{P}(i-1) \mathbf{u}(i) \\ \mathbf{k}(i) &= \beta^{-1} \mathbf{P}(i-1) \mathbf{u}(i) / r(i) \\ e(i) &= d(i) - \mathbf{u}(i)^T \mathbf{w}(i-1) \end{aligned} \tag{4.15}$$

and repeat the argument in the previous section, we obtain the exponentially weighted RLS (EW-RLS) algorithm, labeled Algorithm 6.

Algorithm 6. The exponentially weighted recursive least-squares algorithm

---

Initialization

$\mathbf{w}(0) = 0$, $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$

Computation

Iterate for $i \geq 1$

$$\begin{aligned} r(i) &= 1 + \beta^{-1} \mathbf{u}(i)^T \mathbf{P}(i-1) \mathbf{u}(i) \\ \mathbf{k}(i) &= \beta^{-1} \mathbf{P}(i-1) \mathbf{u}(i) / r(i) \\ e(i) &= d(i) - u(i)^T \mathbf{w}(i-1) \\ \mathbf{w}(i) &= \mathbf{w}(i-1) + \mathbf{k}(i) e(i) \\ \mathbf{P}(i) &= \left[\beta^{-1} \mathbf{P}(i-1) - \mathbf{k}(i) \mathbf{k}(i)^T r(i) \right] \end{aligned} \tag{4.16}$$

4.3 KERNEL RECURSIVE LEAST-SQUARES ALGORITHM

To derive RLS in reproducing kernel Hilbert spaces (RKHS), we use the Mercer theorem to transform the data $\mathbf{u}(i)$ into the feature space $\mathbb{F}$ as $\varphi(\mathbf{u}(i))$

[denoted as $\boldsymbol{\varphi}(i)$].We formulate the recursive least-squares algorithm on the example sequence $\{d(1), d(2), \ldots\}$ and $\{\boldsymbol{\varphi}(1), \boldsymbol{\varphi}(2), \ldots\}$. At each iteration, the weight vector $\boldsymbol{\omega}(i)$, which is the minimizer of

$$\min_{\boldsymbol{\omega}} \sum_{j=1}^{i} |d(j) - \boldsymbol{\omega}^T \boldsymbol{\varphi}(j)|^2 + \lambda \|\boldsymbol{\omega}\|^2 \tag{4.17}$$

needs to be solved recursively as in equation (4.16). As RKHS is a high-dimensional space, regularization here is necessary. Also, because the dimensionality of $\boldsymbol{\varphi}(j)$ is so high (can be even infinite), equation (4.16) cannot be applied directly here, and the solution is not feasible in practice.

By introducing

$$\begin{aligned} \mathbf{d}(i) &= [d(1), \ldots, d(i)]^T \\ \boldsymbol{\Phi}(i) &= [\boldsymbol{\varphi}(1), \ldots, \boldsymbol{\varphi}(i)] \end{aligned} \tag{4.18}$$

we have

$$\boldsymbol{\omega}(i) = \left[\lambda \mathbf{I} + \boldsymbol{\Phi}(i)\boldsymbol{\Phi}(i)^T\right]^{-1} \boldsymbol{\Phi}(i)\mathbf{d}(i) \tag{4.19}$$

Furthermore, using the matrix inversion lemma [equation (4.4)] with the identifications

$$\lambda \mathbf{I} \to \mathbf{A}, \quad \boldsymbol{\Phi}(i) \to \mathbf{B}, \quad \mathbf{I} \to \mathbf{C}, \quad \boldsymbol{\Phi}(i)^T \to \mathbf{D}$$

it is easy to verify that

$$\left[\lambda \mathbf{I} + \boldsymbol{\Phi}(i)\boldsymbol{\Phi}(i)^T\right]^{-1} \boldsymbol{\Phi}(i) = \boldsymbol{\Phi}(i)\left[\lambda \mathbf{I} + \boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i)\right]^{-1}$$

Substituting this result into equation (4.19), we obtain

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)\left[\lambda \mathbf{I} + \boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i)\right]^{-1} \mathbf{d}(i) \tag{4.20}$$

We have to emphasize the significance of the change from equation (4.19) to equation (4.20) here. First, $\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i)$ is computable by the kernel trick [equation (1.28)], and second, the weight is expressed explicitly as a linear combination of the input data:

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)\mathbf{a}(i)$$

with

$$\mathbf{a}(i) = \left[\lambda \mathbf{I} + \boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i)\right]^{-1} \mathbf{d}(i)$$

Denote

$$\mathbf{Q}(i) = \left[\lambda \mathbf{I} + \mathbf{\Phi}(i)^T \mathbf{\Phi}(i)\right]^{-1} \tag{4.21}$$

It is easy to observe that

$$\mathbf{Q}(i)^{-1} = \begin{bmatrix} \mathbf{Q}(i-1)^{-1} & \mathbf{h}(i) \\ \mathbf{h}(i)^T & \lambda + \boldsymbol{\varphi}(i)^T \boldsymbol{\varphi}(i) \end{bmatrix} \tag{4.22}$$

where $\mathbf{h}(i) = \mathbf{\Phi}(i-1)^T \boldsymbol{\varphi}(i)$. Using this sliding-window structure, the updating of the inverse of this growing matrix can be efficient, as shown by

$$\mathbf{Q}(i) = r(i)^{-1} \begin{bmatrix} \mathbf{Q}(i-1) r(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\mathbf{z}(i) \\ -\mathbf{z}(i)^T & 1 \end{bmatrix} \tag{4.23}$$

where

$$\begin{aligned} \mathbf{z}(i) &= \mathbf{Q}(i-1)\mathbf{h}(i) \\ r(i) &= \lambda + \boldsymbol{\varphi}(i)^T \boldsymbol{\varphi}(i) - \mathbf{z}(i)^T \mathbf{h}(i) \end{aligned} \tag{4.24}$$

Equation (4.23) can be established by using the block matrix inversion identity

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \left(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1} & -\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}\left(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1} & \left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1} \end{bmatrix} \tag{4.25}$$

where $\mathbf{A}$ and $\mathbf{D}$ are arbitrary square matrix blocks.

Therefore, the expansion coefficients of the weight of equation (4.20) are

$$\begin{aligned} \mathbf{a}(i) &= \mathbf{Q}(i)\mathbf{d}(i) \\ &= \begin{bmatrix} \mathbf{Q}(i-1) + \mathbf{z}(i)\mathbf{z}(i)^T r(i)^{-1} & -\mathbf{z}(i) r(i)^{-1} \\ -\mathbf{z}(i)^T r(i)^{-1} & r(i)^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{d}(i-1) \\ d(i) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{a}(i-1) - \mathbf{z}(i) r(i)^{-1} e(i) \\ r(i)^{-1} e(i) \end{bmatrix} \end{aligned} \tag{4.26}$$

where $e(i)$ is the prediction error computed by the difference between the desired signal and the prediction $f_{i-1}(\mathbf{u}(i))$:

$$f_{i-1}(\mathbf{u}(i)) = h(i)^T \mathbf{a}(i-1) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa(\mathbf{u}(j), \mathbf{u}(i)) \tag{4.27}$$

$$e(i) = d(i) - f_{i-1}(\mathbf{u}(i)) \tag{4.28}$$

To summarize, KRLS assumes a radial-basis function network structure at each iteration. $\mathbf{a}_j(i-1)$ is the jth component of $\mathbf{a}(i-1)$ in equation (4.27).

From equation (4.26), we observe that the learning procedure of KRLS is similar to KLMS and the kernel affine projection algorithm (KAPA) in the sense that it allocates a new unit with $\mathbf{u}(i)$ as the center and $r(i)^{-1}e(i)$ as the coefficient. At the same time, KRLS also updates all the previous coefficients by $-\mathbf{z}(i)r(i)^{-1}e(i)$, whereas KLMS never updates previous coefficients, and KAPA only updates the $K-1$ most recent ones.

If we denote f_i as the estimate of the input–output mapping at iteration i, then we have the following sequential learning rule for KRLS:

$$f_i = f_{i-1} + r(i)^{-1}\left[\kappa(\mathbf{u}(i),\cdot) - \sum_{j=1}^{i-1}\mathbf{z}_j(i)\kappa(\mathbf{u}(j),\cdot)\right]e(i) \tag{4.29}$$

The coefficients $\mathbf{a}(i)$ and the centers $\mathcal{C}(i)$ should be stored in a computer during training. The updates needed for KRLS at iteration i are

$$\mathbf{a}_i(i) = r(i)^{-1}e(i) \tag{4.30}$$

$$\mathbf{a}_j(i) = \mathbf{a}_j(i-1) - r(i)^{-1}e(i)\mathbf{z}_j(i), \quad j = 1,\cdots,i-1 \tag{4.31}$$

$$\mathcal{C}(i) = \{\mathcal{C}(i-1), \mathbf{u}(i)\} \tag{4.32}$$

KRLS is summarized in Algorithm 7. The time and space complexities are both $O(i^2)$. As we will observe, with some sparsification, the complexity will be reduced to $O(m_i^2)$, where m_i is the effective number of centers in the network at time i.

Algorithm 7. The kernel recursive least-squares algorithm

Initialization

$\mathbf{Q}(1) = (\lambda + \kappa(\mathbf{u}(1),\mathbf{u}(1)))^{-1}$, $\mathbf{a}(1) = \mathbf{Q}(1)d(1)$

Computation

Iterate for $i > 1$:

$$\begin{aligned}
\mathbf{h}(i) &= [\kappa(\mathbf{u}(i),\mathbf{u}(1)),\cdots,\kappa(\mathbf{u}(i),\mathbf{u}(i-1))]^T \\
\mathbf{z}(i) &= \mathbf{Q}(i-1)\mathbf{h}(i) \\
r(i) &= \lambda + \kappa(\mathbf{u}(i),\mathbf{u}(i)) - \mathbf{z}(i)^T\mathbf{h}(i) \\
\mathbf{Q}(i) &= r(i)^{-1}\begin{bmatrix} \mathbf{Q}(i-1)r(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\mathbf{z}(i) \\ -\mathbf{z}(i)^T & 1 \end{bmatrix} \\
e(i) &= d(i) - \mathbf{h}(i)^T\mathbf{a}(i-1) \\
\mathbf{a}(i) &= \begin{bmatrix} \mathbf{a}(i-1) - \mathbf{z}(i)r(i)^{-1}e(i) \\ r(i)^{-1}e(i) \end{bmatrix}
\end{aligned} \tag{4.33}$$

At iteration i, given a test input $\mathbf{u}_*$, the output of the system is

$$f(\mathbf{u}_*) = \sum_{j=1}^{i}\mathbf{a}_j(i)\kappa(\mathbf{u}(j),\mathbf{u}_*) \tag{4.34}$$

4.4 APPROXIMATE LINEAR DEPENDENCY

The implementation of KRLS is straightforward.We need to store $\mathbf{a}(i)$ and $\mathbf{Q}(i)$, so the total time and space complexities are both $O(i^2)$ at iteration i. However, the network size increases linearly with the number of training data as in KLMS and KAPA. The novelty criterion can be used readily without any modification. Here, we discuss another criterion called approximate linear dependency (ALD) proposed by Engel et al. [2004]. Suppose the present dictionary is $\mathcal{C}(i) = \{\mathbf{c}_j\}_{j=1}^{m_i}$, where $\mathbf{c}_j$ is the jth center and m_i is the cardinality. When a new data pair $\{\mathbf{u}(i+1), d(i+1)\}$ is presented, ALD tests the following cost:

$$\text{dis}_2 = \min_{\forall \mathbf{b}} \left\| \boldsymbol{\varphi}(\mathbf{u}(i+1)) - \sum_{\mathbf{c}_j \in \mathcal{C}(i)} \mathbf{b}_j \boldsymbol{\varphi}(\mathbf{c}_j) \right\|$$

which indicates the distance of the new input to the linear span of the present dictionary in the feature space. By straightforward calculus, it turns out that

$$\mathbf{G}(i)\mathbf{b} = \mathbf{h}(i+1) \tag{4.35}$$

where $\mathbf{G}(i)$ and $\mathbf{h}(i+1)$ are redefined based on $\boldsymbol{C}(i)$ as

$$\mathbf{h}(i+1) = [\kappa(\mathbf{u}(i+1), \mathbf{c}_1), \cdots, \kappa(\mathbf{u}(i+1), \mathbf{c}_{m_i})]^T \tag{4.36}$$

$$\mathbf{G}(i) = \begin{bmatrix} \kappa(\mathbf{c}_1, \mathbf{c}_1) & \cdots & \kappa(\mathbf{c}_{m_i}, \mathbf{c}_1) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{c}_1, \mathbf{c}_{m_i}) & \cdots & \kappa(\mathbf{c}_{m_i}, \mathbf{c}_{m_i}) \end{bmatrix} \tag{4.37}$$

Assuming that $\mathbf{G}(i)$ is invertible, we may write

$$\mathbf{b} = \mathbf{G}(i)^{-1}\mathbf{h}(i+1) \tag{4.38}$$

$$\text{dis}_2^2 = \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}(i+1)^T \mathbf{G}^{-1}(i)\mathbf{h}(i+1) \tag{4.39}$$

$\mathbf{u}(i+1)$ will be rejected if dis_2 is smaller than some preset threshold δ_3 in ALD. The computational complexity is $O(m_i^2)$.

However, by equation (4.24), we have

$$r(i+1) = \lambda + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}(i+1)^T (\mathbf{G}(i) + \lambda \mathbf{I})^{-1} \mathbf{h}(i+1) \tag{4.40}$$

in KRLS [equation (4.33)]. On the one hand, note that when λ is small, $r(i+1)$ is essentially equivalent to dis_2. On the other hand, in the Gaussian processes theory, $r(i+1)$ denotes the prediction variance and data selection based on $r(i+1)$ is well documented [Csato and Opper, 2002]. In other words, $r(i+1)$ can be used to approximate ALD and has a solid basis to be used as a data-selection criterion.

Although ALD can be used directly in KRLS, ALD itself is computationally expensive scaling quadratically with the size of the dictionary. A natural simplification is to use the "nearest" center in the dictionary to estimate the overall distance, i.e.,

$$\text{dis}_3 = \min_{\forall b, \forall \mathbf{c}_j \in \mathcal{C}(i)} \left\| \boldsymbol{\varphi}(\mathbf{u}(i+1)) - b\boldsymbol{\varphi}(\mathbf{c}_j) \right\| \tag{4.41}$$

Using straightforward calculus, we may show that

$$\text{dis}_3^2 = \min_{\forall \mathbf{c}_j \in \mathcal{C}(i)} \left[\kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \frac{\kappa^2(\mathbf{u}(i+1), \mathbf{c}_j)}{\kappa(\mathbf{c}_j, \mathbf{c}_j)} \right] \tag{4.42}$$

When κ is a unit-norm kernel, that is, $\kappa(\mathbf{u}, \mathbf{u}) = 1$ for all $\mathbf{u}$, this distance measure is equivalent to the coherence measure discussed in Richard et al. [2009]. When κ is a radial-basis function, this distance measure is equivalent to dis_1 used in the novelty criterion. This understanding gives us a way to estimate δ_3. Assuming the kernel to be Gaussian, we have

$$\begin{aligned} \text{dis}_2 &= \min_{\forall \mathbf{b}} \left\| \boldsymbol{\varphi}(\mathbf{u}(i+1)) - \sum_{\mathbf{c}_j \in \mathcal{C}(i)} \mathbf{b}_j \boldsymbol{\varphi}(\mathbf{c}_j) \right\| \\ &\le \left\| \boldsymbol{\varphi}(\mathbf{u}(i+1)) - \boldsymbol{\varphi}(\mathbf{c}_{nearest}) \right\| \\ &= \left(2 - 2\kappa(\mathbf{u}(i+1), \mathbf{c}_{nearest})\right)^{1/2} \\ &= \left(2 - 2\exp\left(-a\|\mathbf{u}(i+1) - \mathbf{c}_{nearest}\|^2\right)\right)^{1/2} \end{aligned}$$

Assuming δ_1 is the threshold used in the novelty criterion, we have

$$\delta_3 \le \left(2 - 2\exp\left(-a\delta_1^2\right)\right)^{1/2} \approx \sqrt{2a}\delta_1 = \delta_1 / h \tag{4.43}$$

where $h = 1/\sqrt{2a}$ is the kernel bandwidth of the Gaussian kernel. If we pick $\delta_1 = h/10$, then $\delta_3 \le 0.1$.

The original derivation of KRLS in Engel et al. [2004] does not employ regularization, i.e., $\lambda = 0$ in Algorithm 7.[1] The question is how it avoids numerical instability without regularization. On the one hand, if we examine the recursion equation, it is clear that without regularization (or equivalently, $\lambda = 0$), $r(i)$ could be close to 0 and thus cause numerical problems. On the other hand, ALD enforces that $r(i) > \delta_3 > 0$. In this sense, ALD not only is an effective approach to sparsification but also improves the overall stability of the algorithm. See Appendix B for details.

4.5 EXPONENTIALLY WEIGHTED KERNEL RECURSIVE LEAST-SQUARES ALGORITHM

Similarly, we may introduce a forgetting factor in KRLS to deemphasize data from the remote past and regularization. The weighted cost function is

$$\min_{\boldsymbol{\omega}} \sum_{j=1}^{i} \beta^{i-j} \left| d(j) - \boldsymbol{\omega}^T \boldsymbol{\varphi}(j) \right|^2 + \beta^i \lambda \|\boldsymbol{\omega}\|^2 \tag{4.44}$$

and its solution becomes

$$\boldsymbol{\omega}(i) = \left[\lambda \beta^i \mathbf{I} + \boldsymbol{\Phi}(i) \mathbf{B}(i) \boldsymbol{\Phi}(i)^T \right]^{-1} \boldsymbol{\Phi}(i) \mathbf{B}(i) \mathbf{d}(i) \tag{4.45}$$

Then, using the matrix inversion lemma [equation (4.4)] again with the identifications

$$\lambda \beta^i \mathbf{I} \to \mathbf{A}, \quad \boldsymbol{\Phi}(i) \to \mathbf{B}, \quad \mathbf{B}(i) \to \mathbf{C}, \quad \boldsymbol{\Phi}(i)^T \to \mathbf{D}$$

we get

$$\left[\lambda \beta^i \mathbf{I} + \boldsymbol{\Phi}(i) \mathbf{B}(i) \boldsymbol{\Phi}(i)^T \right]^{-1} \boldsymbol{\Phi}(i) \mathbf{B}(i) = \boldsymbol{\Phi}(i) \left[\lambda \beta^i \mathbf{B}(i)^{-1} + \boldsymbol{\Phi}(i)^T \boldsymbol{\Phi}(i) \right]^{-1}$$

Substituting this result into equation (4.45) yields

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i) \left[\lambda \beta^i \mathbf{B}(i)^{-1} + \boldsymbol{\Phi}(i)^T \boldsymbol{\Phi}(i) \right]^{-1} \mathbf{d}(i) \tag{4.46}$$

Now, the weight is expressed explicitly as a linear combination of the input data $\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)\mathbf{a}(i)$ and the coefficients vector

$$\mathbf{a}(i) = \left[\lambda \beta^i \mathbf{B}(i)^{-1} + \boldsymbol{\Phi}(i)^T \boldsymbol{\Phi}(i) \right]^{-1} \mathbf{d}(i)$$

is computable using the kernel trick. Denote

$$\mathbf{Q}(i) = \left[\lambda \beta^i \mathbf{B}(i)^{-1} + \boldsymbol{\Phi}(i)^T \boldsymbol{\Phi}(i) \right]^{-1} \tag{4.47}$$

It is easy to observe that

$$\mathbf{Q}(i)^{-1} = \begin{bmatrix} \mathbf{Q}(i-1)^{-1} & \mathbf{h}(i) \\ \mathbf{h}(i)^T & \lambda \beta^i + \boldsymbol{\varphi}(i)^T \boldsymbol{\varphi}(i) \end{bmatrix} \tag{4.48}$$

where $\mathbf{h}(i) = \boldsymbol{\Phi}(i-1)^{\mathrm{T}} \boldsymbol{\varphi}(i)$. The update rule for the inverse of this growing matrix is

$$\mathbf{Q}(i) = r(i)^{-1} \begin{bmatrix} \mathbf{Q}(i-1) r(i) + \mathbf{z}(i) z(i)^T & -\mathbf{z}(i) \\ -\mathbf{z}(i)^T & 1 \end{bmatrix} \tag{4.49}$$

where

$$\begin{aligned} z(i) &= \mathbf{Q}(i-1) \mathbf{h}(i) \\ r(i) &= \lambda \beta^i + \boldsymbol{\varphi}(i)^T \boldsymbol{\varphi}(i) - \mathbf{z}(i)^T \mathbf{h}(i) \end{aligned} \tag{4.50}$$

Therefore, the expansion coefficients of the weight of equation (4.46) are as follows:

$$\begin{aligned}\mathbf{a}(i) &= \mathbf{Q}(i)\mathbf{d}(i) \\ &= \begin{bmatrix} \mathbf{Q}(i-1)+\mathbf{z}(i)\mathbf{z}(i)^T r(i)^{-1} & -\mathbf{z}(i)r(i)^{-1} \\ -\mathbf{z}(i)^T r(i)^{-1} & r(i)^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{d}(i-1) \\ d(i) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{a}(i-1)-\mathbf{z}(i)r(i)^{-1}e(i) \\ r(i)^{-1}e(i) \end{bmatrix}\end{aligned} \tag{4.51}$$

which is the same as in KRLS.

To summarize, the only difference between exponentially weighted KRLS (EW-KRLS) and KRLS is in the computation of $r(i)$ where the regularization parameter is exponentially weighted as time progresses. EW-KRLS is summarized in Algorithm 8. The time and space complexities are both $O(i^2)$. With sparsification, the complexity will be reduced to $O(m_i^2)$, where m_i is the effective number of centers in the network at time i. ALD still works here by using the new $r(i)$ directly.

Algorithm 8. The exponentially weighted kernel recursive least-squares algorithm

Initialization

$\mathbf{Q}(1) = (\lambda\beta + \kappa(\mathbf{u}(1), \mathbf{u}(1))^{-1}$, $\mathbf{a}(1) = \mathbf{Q}(1)d(1)$

Computation

Iterate for $i > 1$:

$$\begin{aligned}\mathbf{h}(i) &= [\kappa(\mathbf{u}(i), \mathbf{u}(1)), \cdots, \kappa(\mathbf{u}(i), \mathbf{u}(i-1))]^T \\ \mathbf{z}(i) &= \mathbf{Q}(i-1)\mathbf{h}(i) \\ r(i) &= \lambda\beta^i + \kappa(\mathbf{u}(i), \mathbf{u}(i)) - \mathbf{z}(i)^T\mathbf{h}(i) \\ \mathbf{Q}(i) &= r(i)^{-1}\begin{bmatrix} \mathbf{Q}(i-1)r(i)+\mathbf{z}(i)\mathbf{z}(i)^T & -\mathbf{z}(i) \\ -\mathbf{z}(i)^T & 1 \end{bmatrix} \\ e(i) &= d(i) - \mathbf{h}(i)^T\mathbf{a}(i-1) \\ \mathbf{a}(i) &= \begin{bmatrix} \mathbf{a}(i-1)-\mathbf{z}(i)r(i)^{-1}e(i) \\ r(i)^{-1}e(i) \end{bmatrix}\end{aligned} \tag{4.52}$$

4.6 GAUSSIAN PROCESSES FOR LINEAR REGRESSION

GPR[2] provides a nice Bayesian interpretation to the least-squares problem we just discussed in the previous sections. We start with least squares in the linear space then discuss the general case by projecting the input into a high-dimensional feature space. Suppose we have a training set of i observations, $\mathcal{D}(i) = \{\mathbf{u}(j), d(j)\}_{j=1}^{i}$ where $\mathbf{u}(j)$ is the input of dimension L and $d(j)$ is the

scalar desired or target. And we assume the underlying relationship between the input and the desired is a standard linear regression model with Gaussian noise

$$f(\mathbf{u}) = \mathbf{u}^T\mathbf{w}, \quad d = f(\mathbf{u}) + v \tag{4.53}$$

where $\mathbf{w}$ is the weight vector of the linear model and f is the function and d is the observed target value. v is the additive noise that follows an independent, identically distributed Gaussian distribution with zero mean and variance σ_n^2. With all these assumptions, the *likelihood* of the observations given the inputs and weight vector is

$$\begin{aligned} p(\mathbf{d}(i)|\mathbf{U}(i),\mathbf{w}) &= \prod_{j=1}^{i} p(d(j)|\mathbf{u}(j),\mathbf{w}) \\ &= \prod_{j=1}^{i} \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{\left(d(j)-\mathbf{u}(j)^T\mathbf{w}\right)^2}{2\sigma_n^2}\right) \\ &= \frac{1}{\left(2\pi\sigma_n^2\right)^{i/2}} \exp\left(-\frac{\sum_{j=1}^{i}\left(d(j)-\mathbf{u}(j)^T\mathbf{w}\right)^2}{2\sigma_n^2}\right) \\ &= \frac{1}{\left(2\pi\sigma_n^2\right)^{i/2}} \exp\left(-\frac{\left\|\mathbf{d}(i)-\mathbf{U}(i)^T\mathbf{w}\right\|^2}{2\sigma_n^2}\right) \\ &= \mathcal{N}\left(\mathbf{U}(i)^T\mathbf{w}, \sigma_n^2\mathbf{I}\right) \end{aligned} \tag{4.54}$$

with

$$\mathbf{U}(i) = [\mathbf{u}(1),\ldots,\mathbf{u}(i)]_{L\times i} \quad \text{and} \quad \mathbf{d}(i) = [d(1),\ldots,d(i)]^T$$

Bayes' rule states that

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

or, in this particular case,

$$p(\mathbf{w}|\mathbf{U}(i),\mathbf{d}(i)) = \frac{p(\mathbf{d}(i)|\mathbf{U}(i),\mathbf{w}) \times p(\mathbf{w})}{p(\mathbf{d}(i)|\mathbf{U}(i))} \tag{4.55}$$

Clearly, to derive the posterior distribution over the weight vector, we need a *prior* over the weight vector. We assume a zero-mean Gaussian prior with covariance matrix $\sigma_{\mathbf{w}}^2\mathbf{I}$

$$p(\mathbf{w}) = \mathcal{N}\left(\mathbf{0}, \sigma_{\mathbf{w}}^2\mathbf{I}\right)$$

where $\mathbf{I}$ is the L-by-L identity matrix. The denominator of equation (4.55) is known as the evidence, or the marginal likelihood, which is independent of $\mathbf{w}$ and given by

$$p(\mathbf{d}(i)|\mathbf{U}(i)) = \int [p(\mathbf{d}(i)|\mathbf{U}(i), \mathbf{w}) \times p(\mathbf{w})] d\mathbf{w} \tag{4.56}$$

Because the marginal likelihood is merely a normalizing constant, we may derive the posterior as

$$\begin{aligned} p(\mathbf{w}|\mathbf{U}(i), \mathbf{d}(i)) &\propto p(\mathbf{d}(i)|\mathbf{U}(i), \mathbf{w}) \times p(\mathbf{w}) \\ &\propto \exp\left(-\frac{\left\| \mathbf{d}(i) - \mathbf{U}(i)^T \mathbf{w} \right\|^2}{2\sigma_n^2} \right) \exp\left(-\frac{1}{2} \mathbf{w}^T \left(\sigma_{\mathbf{w}}^2 \mathbf{I} \right)^{-1} \mathbf{w} \right) \\ &\propto \exp\left[-\frac{1}{2} (\mathbf{w} - \mathbf{w}(i))^T \left(\frac{1}{\sigma_n^2} \mathbf{U}(i) \mathbf{U}(i)^T + \sigma_{\mathbf{w}}^{-2} \mathbf{I} \right) (\mathbf{w} - \mathbf{w}(i)) \right] \end{aligned} \tag{4.57}$$

where

$$\mathbf{w}(i) = \left(\mathbf{U}(i) \mathbf{U}(i)^T + \sigma_n^2 \sigma_{\mathbf{w}}^{-2} \mathbf{I} \right)^{-1} \mathbf{U}(i) \mathbf{d}(i) \tag{4.58}$$

It is clear from equation (4.57) that the posterior distribution is Gaussian with mean $\mathbf{w}(i)$ and covariance matrix $\left(\frac{1}{\sigma_n^2} \mathbf{U}(i) \mathbf{U}(i)^T + \sigma_{\mathbf{w}}^{-2} \mathbf{I} \right)^{-1}$.

For any Gaussian posterior, the mean is also the *mode*. Hence, $\mathbf{w}(i)$ is called the maximum a posteriori (MAP) estimate of $\mathbf{w}$. Comparing equation (4.58) with equation (4.11), we observe that they are equivalent if

$$\lambda = \sigma_n^2 \sigma_{\mathbf{w}}^{-2}$$

i.e., large noise variance and strong prior amount to strong regularization. This equivalency is well known in the literature. Also, we have to emphasize that if there are sufficient data, the effect of the prior diminishes from the viewpoint of Bayesian inference.

To make a prediction for a test point $\mathbf{u}_*$, we have the posterior distribution for $f(\mathbf{u}_*)$

$$p(f(\mathbf{u}_*)|\mathbf{u}_*, \mathcal{D}(i)) = \int p(f(\mathbf{u}_*)|\mathbf{u}_*, \mathbf{w}) \, p(\mathbf{w}|\mathcal{D}(i)) d\mathbf{w} \tag{4.59}$$

which is a weighted average by the posterior distribution over all possible weight. The posterior distribution of the prediction is again Gaussian, with mean $\mathbf{u}_*^T \mathbf{w}(i)$ and variance $\mathbf{u}_*^T \left(\frac{1}{\sigma_n^2} \mathbf{U}(i) \mathbf{U}(i)^T + \sigma_{\mathbf{w}}^{-2} \mathbf{I} \right)^{-1} \mathbf{u}_*$.

The formula of the prediction mean coincides with the result in linear regression. The formula of the prediction variance is a quadratic form of the

test input with the posterior covariance matrix. The predictive uncertainty is large if 1) the magnitude of the test input is large or 2), the direction of the test input aligns with the eigenvector of $\left(\frac{1}{\sigma_n^2}\mathbf{U}(i)\mathbf{U}(i)^T+\sigma_{\mathbf{w}}^{-2}\mathbf{I}\right)^{-1}$ associated with the largest eigenvalue. If we assume that the posterior covariance matrix is dominated by $\mathbf{U}(i)\mathbf{U}(i)^T$, then the eigenvector of $\left(\frac{1}{\sigma_n^2}\mathbf{U}(i)\mathbf{U}(i)^T+\sigma_{\mathbf{w}}^{-2}\mathbf{I}\right)^{-1}$ for the largest eigenvalue comes from the eigenvector of $\mathbf{U}(i)\mathbf{U}(i)^T$ for the smallest eigenvalue. This observation is intuitively satisfying because it means the prediction uncertainty is large if the test input points in the direction of the minor component of the existing data; in other words, the prediction uncertainty is small if the test input points in the direction of the principal component of the existing data.

4.7 GAUSSIAN PROCESSES FOR NONLINEAR REGRESSION

Just as we derived kernel adaptive filtering techniques, we assume a standard linear regression model in RKHS, setting

$$f(\mathbf{u})=\boldsymbol{\varphi}(\mathbf{u})^T\boldsymbol{\omega},\quad d=f(\mathbf{u})+v \tag{4.60}$$

where $\boldsymbol{\varphi}(\mathbf{u})$ is the transformed input in the feature space induced by a kernel $\tilde{\kappa}$, $\boldsymbol{\omega}$ is the weight vector of the linear model in the feature space, f is the function, and d is the observed target value. v is the additive noise that follows an independent, identically distributed Gaussian distribution with zero mean and variance σ_n^2. With all these assumptions, the *likelihood* of the observations given the inputs and weight vector can be similarly derived as shown by

$$\begin{aligned}
p(\mathbf{d}(i)|\mathbf{U}(i),\boldsymbol{\omega}) &= p(\mathbf{d}(i)|\boldsymbol{\Phi}(i),\boldsymbol{\omega})\\
&=\prod_{j=1}^{i}p(d(j)|\boldsymbol{\varphi}(\mathbf{u}(j)),\boldsymbol{\omega})\\
&=\prod_{j=1}^{i}\frac{1}{\sqrt{2\pi\sigma_n^2}}\exp\left(-\frac{\left(d(j)-\boldsymbol{\varphi}(\mathbf{u}(j))^T\boldsymbol{\omega}\right)^2}{2\sigma_n^2}\right)\\
&=\frac{1}{\left(2\pi\sigma_n^2\right)^{i/2}}\exp\left(-\frac{\sum_{j=1}^{i}\left(d(j)-\boldsymbol{\varphi}(\mathbf{u}(j))^T\boldsymbol{\omega}\right)^2}{2\sigma_n^2}\right)\\
&=\frac{1}{\left(2\pi\sigma_n^2\right)^{i/2}}\exp\left(-\frac{\left\|\mathbf{d}(i)-\boldsymbol{\Phi}(i)^T\boldsymbol{\omega}\right\|^2}{2\sigma_n^2}\right)\\
&=\mathcal{N}\left(\boldsymbol{\Phi}(i)^T\boldsymbol{\omega},\sigma_n^2\mathbf{I}\right)
\end{aligned} \tag{4.61}$$

with

$$\mathbf{\Phi}(i) = [\boldsymbol{\varphi}(\mathbf{u}(1)), \ldots, \boldsymbol{\varphi}(\mathbf{u}(i))] \quad \text{and} \quad \mathbf{d}(i) = [d(1), \ldots, d(i)]^T$$

Furthermore, we assume a zero-mean Gaussian prior for $\boldsymbol{\omega}$ with covariance matrix $\sigma_{\omega}^2\mathbf{I}$:

$$p(\boldsymbol{\omega}) = \mathcal{N}(0, \sigma_{\omega}^2\mathbf{I})$$

Therefore, we have the posterior

$$\begin{aligned} p(\boldsymbol{\omega}|\mathbf{U}(i), \mathbf{d}(i)) &\propto p(\mathbf{d}(i)|\mathbf{U}(i), \boldsymbol{\omega}) \times p(\boldsymbol{\omega}) \\ &\propto \exp\left(-\frac{\left\|\mathbf{d}(i) - \mathbf{\Phi}(i)^T\boldsymbol{\omega}\right\|^2}{2\sigma_n^2}\right)\exp\left(-\frac{1}{2}\boldsymbol{\omega}^T(\sigma_{\omega}^2\mathbf{I})^{-1}\boldsymbol{\omega}\right) \\ &\propto \exp\left[-\frac{1}{2}(\boldsymbol{\omega} - \boldsymbol{\omega}(i))^T\left(\frac{1}{\sigma_n^2}\mathbf{\Phi}(i)\mathbf{\Phi}(i)^T + \sigma_{\omega}^{-2}\mathbf{I}\right)(\boldsymbol{\omega} - \boldsymbol{\omega}(i))\right] \end{aligned} \quad (4.62)$$

where

$$\boldsymbol{\omega}(i) = \left(\mathbf{\Phi}(i)\mathbf{\Phi}(i)^T + \sigma_n^2\sigma_{\omega}^{-2}\mathbf{I}\right)^{-1}\mathbf{\Phi}(i)\mathbf{d}(i) \quad (4.63)$$

It is clear from equation (4.62) that the posterior distribution is Gaussian with mean $\boldsymbol{\omega}(i)$ and covariance matrix $\left(\frac{1}{\sigma_n^2}\mathbf{\Phi}(i)\mathbf{\Phi}(i)^T + \sigma_{\omega}^{-2}\mathbf{I}\right)^{-1}$.

Usually, $\boldsymbol{\omega}$ resides in a high-dimensional space, and we cannot deal with it directly. Fortunately, we can still make predictions for any given test point $\mathbf{u}_*$. The posterior distribution for $f(\mathbf{u}_*)$ can be derived similarly as follows:

$$p(f(\mathbf{u}_*)|\mathbf{u}_*, \mathcal{D}(i)) = \int p(f(\mathbf{u}_*)|\mathbf{u}_*, \boldsymbol{\omega})\, p(\boldsymbol{\omega}|\mathcal{D}(i))\, d\boldsymbol{\omega} \quad (4.64)$$

which is Gaussian with mean

$$m(f(\mathbf{u}_*)) = \boldsymbol{\varphi}(\mathbf{u}_*)^T\boldsymbol{\omega}(i)$$

and variance

$$\sigma^2(f(\mathbf{u}_*)) = \boldsymbol{\varphi}(\mathbf{u}_*)^T\left(\frac{1}{\sigma_n^2}\mathbf{\Phi}(i)\mathbf{\Phi}(i)^T + \sigma_{\omega}^{-2}\mathbf{I}\right)^{-1}\boldsymbol{\varphi}(\mathbf{u}_*)$$

Using the matrix inversion lemma, we obtain

$$\begin{aligned} m(f(\mathbf{u}_*)) &= \boldsymbol{\varphi}(\mathbf{u}_*)^T\boldsymbol{\omega}(i) \\ &= \boldsymbol{\varphi}(\mathbf{u}_*)^T\left(\mathbf{\Phi}(i)\mathbf{\Phi}(i)^T + \sigma_n^2\sigma_{\omega}^{-2}\mathbf{I}\right)^{-1}\mathbf{\Phi}(i)\mathbf{d}(i) \\ &= \boldsymbol{\varphi}(\mathbf{u}_*)^T\mathbf{\Phi}(i)\left(\mathbf{\Phi}(i)^T\mathbf{\Phi}(i) + \sigma_n^2\sigma_{\omega}^{-2}\mathbf{I}\right)^{-1}\mathbf{d}(i) \end{aligned} \quad (4.65)$$

and

$$\sigma^2(f(\mathbf{u}_*)) = \boldsymbol{\varphi}(\mathbf{u}_*)^T\left(\frac{1}{\sigma_n^2}\boldsymbol{\Phi}(i)\boldsymbol{\Phi}(i)^T + \sigma_\omega^{-2}\mathbf{I}\right)^{-1}\boldsymbol{\varphi}(\mathbf{u}_*)$$
$$= \sigma_\omega^2\boldsymbol{\varphi}(\mathbf{u}_*)^T\boldsymbol{\varphi}(\mathbf{u}_*) - \sigma_\omega^2\boldsymbol{\varphi}(\mathbf{u}_*)^T\boldsymbol{\Phi}(i)\left(\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \sigma_n^2\sigma_\omega^{-2}\mathbf{I}\right)^{-1}\boldsymbol{\Phi}(i)^T\boldsymbol{\varphi}(\mathbf{u}_*) \tag{4.66}$$

And now all the terms involving the transformed input $\boldsymbol{\varphi}(\mathbf{u}^*)^T\boldsymbol{\Phi}(i), \boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i)$, $\boldsymbol{\varphi}(\mathbf{u}^*)^T\boldsymbol{\varphi}(\mathbf{u}^*)$ and $\boldsymbol{\Phi}(i)^T\boldsymbol{\varphi}(\mathbf{u}^*)$ are computable in the input space by using the kernel trick. Denote

$$\tilde{\mathbf{h}}_* = \boldsymbol{\varphi}(\mathbf{u}_*)^T\boldsymbol{\Phi}(i) = [\tilde{\kappa}(\mathbf{u}_*, \mathbf{u}(1)), \cdots, \tilde{\kappa}(\mathbf{u}_*, \mathbf{u}(i))]^T \tag{4.67}$$

$$\tilde{\mathbf{G}}(i) = \boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) = \begin{bmatrix} \tilde{\kappa}(\mathbf{u}(1), \mathbf{u}(1)) & \cdots & \tilde{\kappa}(\mathbf{u}(i), \mathbf{u}(1)) \\ \vdots & \ddots & \vdots \\ \tilde{\kappa}(\mathbf{u}(1), \mathbf{u}(i)) & \cdots & \tilde{\kappa}(\mathbf{u}(i), \mathbf{u}(i)) \end{bmatrix} \tag{4.68}$$

We obtain the prediction mean as

$$m(f(\mathbf{u}_*)) = \tilde{\mathbf{h}}_*^T\left(\tilde{\mathbf{G}}(i) + \sigma_n^2\sigma_\omega^{-2}\mathbf{I}\right)^{-1}\mathbf{d}(i)$$

and the prediction variance as

$$\sigma^2(f(\mathbf{u}_*)) = \sigma_\omega^2\left[\tilde{\kappa}(\mathbf{u}_*, \mathbf{u}_*) - \tilde{\mathbf{h}}_*^T\left(\tilde{\mathbf{G}}(i) + \sigma_n^2\sigma_\omega^{-2}\mathbf{I}\right)^{-1}\tilde{\mathbf{h}}_*\right]$$

We can simplify the equations even more by absorbing the constant σ_ω^2 into the kernel and defining the new kernel as

$$\kappa = \sigma_\omega^2\tilde{\kappa}$$

This kernel function is also called *covariance function* in the literature, because it is the covariance function of the system output

$$\begin{aligned} \mathbf{E}[f(\mathbf{u})f(\mathbf{u}')] &= \mathbf{E}\left[\boldsymbol{\varphi}(\mathbf{u})^T\boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{\varphi}(\mathbf{u}')\right] \\ &= \boldsymbol{\varphi}(\mathbf{u})^T\mathbf{E}\left[\boldsymbol{\omega}\boldsymbol{\omega}^T\right]\boldsymbol{\varphi}(\mathbf{u}') \\ &= \sigma_\omega^2\tilde{\kappa}(\mathbf{u}, \mathbf{u}') \\ &= \kappa(\mathbf{u}, \mathbf{u}') \end{aligned} \tag{4.69}$$

With the new kernel, the prediction mean and variance, respectively, assume the forms

$$m(f(\mathbf{u}_*)) = \mathbf{h}_*^T\left(\mathbf{G}(i) + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{d}(i) \tag{4.70}$$

$$\sigma^2(f(\mathbf{u}_*)) = \kappa(\mathbf{u}_*, \mathbf{u}_*) - \mathbf{h}_*^T\left(\mathbf{G}(i) + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{h}_* \tag{4.71}$$

with

$$\mathbf{h}_* = [\kappa(\mathbf{u}_*, \mathbf{u}(1)), \cdots, \kappa(\mathbf{u}_*, \mathbf{u}(i))]^T \tag{4.72}$$

$$\mathbf{G}(i) = \begin{bmatrix} \kappa(\mathbf{u}(1), \mathbf{u}(1)) & \cdots & \kappa(\mathbf{u}(i), \mathbf{u}(1)) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{u}(1), \mathbf{u}(i)) & \cdots & \kappa(\mathbf{u}(i), \mathbf{u}(i)) \end{bmatrix} \tag{4.73}$$

This is the posterior distribution of the system output. The output is distorted by additive Gaussian noise, which becomes the actual observation d_*. Because the noise is zero mean, the mean of d_* is the same as the mean of $f(\mathbf{u}_*)$. The variance of d_* is enlarged by the noise variance.

$$m(d_*) = \mathbf{h}_*^T \left(\mathbf{G}(i) + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{d}(i) \tag{4.74}$$

$$\sigma^2(d_*) = \sigma_n^2 + \kappa(\mathbf{u}_*, \mathbf{u}_*) - \mathbf{h}_*^T \left(\mathbf{G}(i) + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{h}_* \tag{4.75}$$

The formula of the prediction mean is actually equivalent to the system output equation in KRLS if we denote

$$\mathbf{a}(i) = \left(\mathbf{G}(i) + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{d}(i) \tag{4.76}$$

and make σ_n^2 equal to λ in KRLS. The formula of the prediction variance is also equivalent to equation (4.40) in KRLS, which is the key quantity in ALD. This observation provides the following interesting findings: On one hand, ALD selects an input as a new center if and only if the system is uncertain about its prediction from the viewpoint of Gaussian process regression. On the other hand, the prediction uncertainty in Gaussian process regression actually measures the distance between the test point and the linear span of the existing centers in the feature space.

4.8 BAYESIAN MODEL SELECTION

Three main approaches to model selection are as follows: 1) estimate and minimize the *generalization error* like in cross-validation, 2) establish and minimize a generalization error upper bound like in statistical learning theory, and 3) estimate and maximize the posterior probability of the model given the data using Bayesian inference. By generalization error, we mean the expected error of the learning system tested on unseen data. In Chapter 2, we discussed the cross-validation method to select the function form of kernel from different families and to specify free parameters within the chosen family. The formulation of cross-validation is model independent and is thus widely applicable. However, when the number of free parameters increases, the computational complexity explodes. Here, we present another useful approach,

which is rooted in Bayesian inference; see, e.g., MacKay [1992b] and Rasmussen and Williams [2006] for this framework. We roughly follow the derivation in Rasmussen and Williams [2006] here.

We use the following hierarchical specification of models. At the top level, we have a set of possible model structures, i.e., the function form of the kernel (Gaussian, Laplacian, polynomial, etc). We use $\mathcal{H}_i$ to denote all possible choices. After we choose the function form of the kernel, at the intermediate level, we need to specify the free parameters in the kernel such as the kernel size in Gaussian kernel, the noise variance, etc. We denote all the free parameters of the chosen kernel as a vector $\boldsymbol{\theta}$. At the bottom level, we have the model coefficients such as $\mathbf{w}$ and $\boldsymbol{\omega}$. It is natural that learning starts from top and ends at bottom. Assume we have N input-output observations $\{\mathbf{u}(j), d(j)\}_{j=1}^{N}$ and denote $\mathbf{U}$ as the input data matrix and $\mathbf{d}$ the corresponding vector of the desired outputs.

At the top level, we compute the *posterior* for the model structure

$$p(\mathcal{H}_i|\mathbf{d}, \mathbf{U}) = \frac{p(\mathbf{d}|\mathbf{U}, \mathcal{H}_i)\, p(\mathcal{H}_i)}{p(\mathbf{d}|\mathbf{U})} \tag{4.77}$$

where $p(\mathbf{d}|\mathbf{U}, \mathcal{H}_i)$ is the top-level model structure *likelihood*, which encodes as a probability distribution the information from the observations, and $p(\mathcal{H}_i)$ is the model structure *prior*. The prior denotes the designers' knowledge about the model structure prior to observing the data. If we only have a vague idea about the model structure, a broad prior distribution is chosen to reflect this. The posterior combines the information from the data and the prior. The denominator $p(\mathbf{d}|\mathrm{U})$ is called the *marginal likelihood* or the *evidence*, which is a normalizing constant independent of $\mathcal{H}_i$ and can be computed as

$$p(\mathbf{d}|\mathbf{U}) = \sum_i p(\mathbf{d}|\mathbf{U}, \mathcal{H}_i)\, p(\mathcal{H}_i)$$

The computation of the top-level likelihood is usually difficult, and this step in inference is commonly skipped.

At the second level, assuming the "best" $\mathcal{H}_i$ is chosen, we have

$$p(\boldsymbol{\theta}|\mathbf{d}, \mathbf{U}, \mathcal{H}_i) = \frac{p(\mathbf{d}|\mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i)\, p(\boldsymbol{\theta}|\mathcal{H}_i)}{p(\mathbf{d}|\mathbf{U}, \mathcal{H}_i)} \tag{4.78}$$

where $p(\mathbf{d}|\mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i)$ is the second-level free parameter likelihood. $p(\boldsymbol{\theta}|\mathcal{H}_i)$ is the parameter prior. The denominator $p(\mathbf{d}|\mathbf{U}, \mathcal{H}_i)$, which is the likelihood at the top level, is given by

$$p(\mathbf{d}|\mathbf{U}, \mathcal{H}_i) = \int p(\mathbf{d}|\mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i) p(\boldsymbol{\theta}|\mathcal{H}_i)\, d\boldsymbol{\theta} \tag{4.79}$$

Fortunately, there is a simple way to compute the second-level likelihood in GPR. Noting that the posterior distribution $\mathbf{d}$ is jointly Gaussian, we obtain

$$\log p(\mathbf{d}|\mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i) = -\frac{1}{2}\mathbf{d}^T\left(\mathbf{G} + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{d} - \frac{1}{2}\log\left|\mathbf{G} + \sigma_n^2\mathbf{I}\right| - \frac{N}{2}\log(2\pi) \tag{4.80}$$

where $\mathbf{G}$ is the Gram matrix

$$G = \begin{bmatrix} \kappa(\mathbf{u}(1), \mathbf{u}(1)) & \cdots & \kappa(\mathbf{u}(N), \mathbf{u}(1)) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{u}(1), \mathbf{u}(N)) & \cdots & \kappa(\mathbf{u}(N), \mathbf{u}(N)) \end{bmatrix} \tag{4.81}$$

and $\left|\mathbf{G} + \sigma_n^2\mathbf{I}\right|$ denotes the determinant of $\mathbf{G} + \sigma_n^2\mathbf{I}$. Then, by assuming appropriate prior of the parameters, we may have the posterior of the parameters. By maximizing the posterior with respect to $\boldsymbol{\theta}$, we have a principled way to select the parameters. In practice, we may instead maximize the likelihood directly with respect to $\boldsymbol{\theta}$ as an approximation.

To summarize, we can use equation (4.80) as an objective function to select the kernel and its free parameters. For each possible kernel (which corresponds to one possible $\mathcal{H}_i$ in the formulation), we maximize the following objective function with respect to $\boldsymbol{\theta}$:

$$J(\mathcal{H}_i) = \max_{\boldsymbol{\theta}}\left[-\frac{1}{2}\mathbf{d}^T\left(\mathbf{G} + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{d} - \frac{1}{2}\log\left|\mathbf{G} + \sigma_n^2\mathbf{I}\right| - \frac{N}{2}\log(2\pi)\right] \tag{4.82}$$

Then, $\mathcal{H}^\circ$, which maximizes $J(\mathcal{H}_i)$, is the best kernel with the corresponding $\boldsymbol{\theta}^\circ$ from the previous optimization. The maximization problem in equation (4.82) can be solved by any standard optimization package like conjugate gradient [Shewchuk, 1994, Golub and Loan, 1996].

At the bottom level, after the optimal $\boldsymbol{\theta}$ and $\mathcal{H}_i$ are chosen, the posterior over the model coefficients is given by

$$p(\mathbf{w}|\mathbf{d}, \mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i) = \frac{p(\mathbf{d}|\mathbf{U}, \mathbf{w}, \boldsymbol{\theta}, \mathcal{H}_i)\, p(\mathbf{w}|\mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i)}{p(\mathbf{d}|\mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i)} \tag{4.83}$$

where $p(\mathbf{d}|\mathbf{U}, \mathbf{w}, \boldsymbol{\theta}, \mathcal{H}_i)$ is the bottom-level model coefficient likelihood and $p(\mathbf{w}|\mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i)$, which equals to $p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)$, is the coefficient prior. The posterior combines the information from the data and the prior. The denominator $p(\mathbf{d}|\mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i)$ is called the bottom-level marginal likelihood, which is a normalizing constant independent of $\mathbf{w}$ and can be computed as

$$p(\mathbf{d}|\mathbf{U}, \boldsymbol{\theta}, \mathcal{H}_i) = \int p(\mathbf{d}|\mathbf{U}, \boldsymbol{\theta}, \mathbf{w}, \mathcal{H}_i)\, p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)\, d\mathbf{w} \tag{4.84}$$

Notice that the bottom-level marginal likelihood is the second-level likelihood we are maximizing, so the methodology described above is called *maximum marginal likelihood*.

Because we have established one-to-one equivalence between KRLS and GPR, the maximum marginal likelihood method can be readily applied to KRLS for kernel design. If we view KLMS and KAPA as stochastic approximations of KRLS, we may argue that an optimal kernel for KRLS is also suitable for other kernel adaptive filters. In other words, the maximum marginal likelihood, which is derived for covariance function design in GPR, can be used for kernel selection in the design of any kernel adaptive filter.

4.9 COMPUTER EXPERIMENTS

4.9.1 KRLS Applied to Mackey–Glass Time-Series Prediction

We use the Mackey–Glass (MG) chaotic time series as a benchmark data set to compare two different sparsification approaches in KRLS: the novelty criterion and the approximate linear dependency test. The problem setting for short-term prediction is as follows: The previous seven points $\mathbf{u}(i) = [x(i-7), x(i-6), \ldots, x(i-1)]^T$ are used to predict the present one $x(i)$. A segment of 1000 samples is used as the training data and another 100 points are used as the test data (in the testing phase, the filter is fixed). The data are corrupted by an additive white Gaussian noise with zero mean and variance 0.001. A Gaussian kernel with kernel parameter $a = 1$ is chosen. The regularization parameter is set 0.001 in KRLS.

Part 1: To have a comprehensive comparison, we choose 20 values of δ_1 uniformly in the interval of [0.05, 0.2] and 20 values of δ_2 uniformly in the interval of [0.05, 0.2] for the novelty criterion (NC). For each possible combination of δ_1 and δ_2, we train the KRLS algorithm with the corresponding NC and record the network size and the testing mean square error (MSE). We also choose 100 values of δ_3 uniformly in the interval of [0.05, 0.3] for ALD. For each δ_3, we train the KRLS algorithm with ALD and record the network size and the testing MSE. Each simulation result is plotted in Figure 4.1 as a point in the two-dimesional space of network size and testing MSE. As we can view, only for small size networks (<80) ALD performs better than NC, whereas NC can actually outperform ALD elsewhere.

Note that ALD is an unsupervised method that does not require the desired signal, whereas NC is a supervised method that uses the desired signal. Also, ALD does not require extra computation in KRLS, whereas NC requires an extra step to compute the distance to the nearest neighbor. An interesting question is what if we also use the error information in ALD just like in NC.

Part 2: Here, we introduce an enhanced novelty criterion (ENC) by combining ALD with NC. First, we check if

$$\lambda + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}(i+1)^T (\mathbf{G}(i) + \lambda \mathbf{I})^{-1} \mathbf{h}(i+1)$$

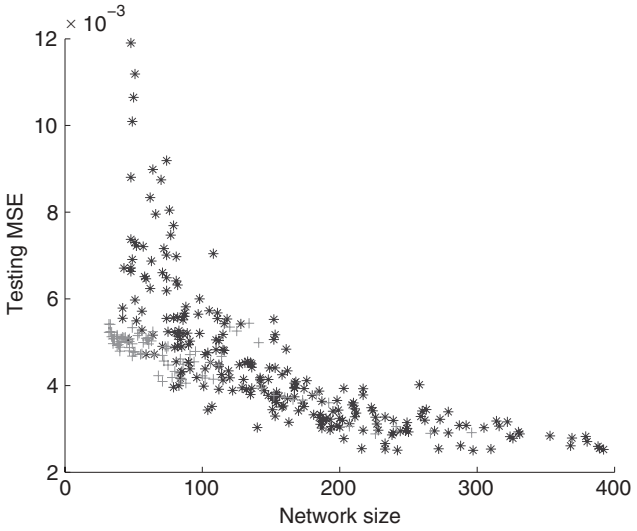


Figure 4.1. Comparison of novelty criterion and approximate linear dependency in KRLS. (Points marked by "*" are from KRLS-NC and points marked by "+" are from KRLS-ALD).

is smaller than δ_3. If it is, we then discard the data. Otherwise, we check the prediction error to determine whether it exceeds the threshold δ_2. Only if the prediction error is larger than δ_2, the input is accepted as a new center.

As in **Part 1**, we choose 20 values of δ_3 uniformly in the interval of [0.04, 0.2] and 20 values of δ_2 uniformly in the interval of [0.05, 0.2] for the enhanced novelty criterion. For each possible combination of δ_3 and δ_2, we train the KRLS algorithm with ENC and record the network size and the testing MSE. The result is plotted in Figure 4.2 along with NC and ALD. As we can observe, the enhanced novelty criterion performs well compared with ALD and NC. It can achieve low testing MSE with a small network size. And the performance boost is from one extra comparison operation.

4.9.2 Model Selection by Maximum Marginal Likelihood

In this example, we show how to use the maximum marginal likelihood to determine the free parameters in GPR and generally any kernel method. We use the Gaussian Process for Machine Learning (GPML) package coded by Rasmussen and Williams. The reader may download the package from the following website: http://www.gaussianprocess.org/gpml/code/matlab/doc/ and add the package path into the work directory of Matlab.

Part 1: In our discussion of GPR, we started with the linear regression model either in the input space or in the feature space. There is actually a more elegant, albeit more abstract way, to reach identical results. A Gaussian

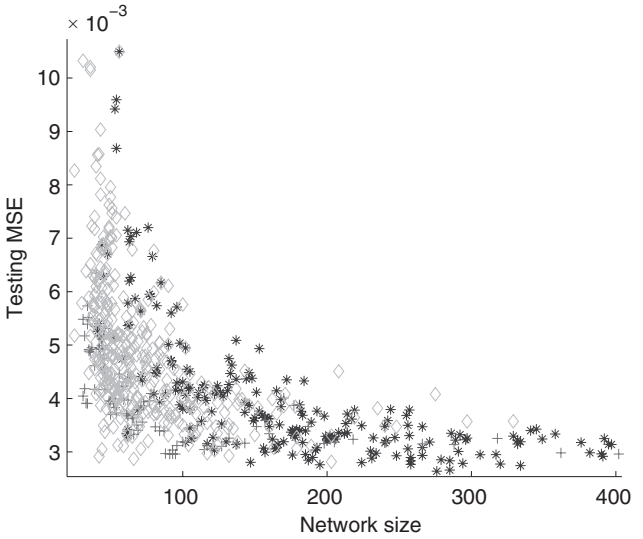


Figure 4.2. Performance of enhanced novelty criterion in KRLS. (Points marked as "diamond" are from KRLS-ENC. Points marked by "*" are from KRLS-NC and points marked by "+" are from KRLS-ALD.)

process is a random process such that any finite number of samples have a joint Gaussian distribution.

Denote $f(\mathbf{x})$ as a Gaussian process. We define the mean function $m(\mathbf{x})$ and the covariance function $\kappa(\mathbf{x}, \mathbf{x}')$ as follows:

$$m(\mathbf{x}) = \mathbf{E}[f(\mathbf{x})] \tag{4.85}$$

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}))] \tag{4.86}$$

The covariance function is a symmetric, positive-definite function just like the reproducing kernel. As a matter of fact, any reproducing kernel is a covariance function and any covariance function is a reproducing kernel.

By definition, given any integer N and any possible combination of samples $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, the joint distribution of $\{f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_N)\}$ is Gaussian with the mean vector

$$[m(\mathbf{x}_1), m(\mathbf{x}_2), \ldots, m(\mathbf{x}_N)]^T$$

and the covariance matrix

$$\begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_1, \mathbf{x}_N) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} \tag{4.87}$$

This observation gives us a way to generate a data set sampled from the Gaussian process. To simplify the discussion, we assume the mean function is zero everywhere. First, we generate N inputs by sampling the input domain, which are denoted by $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. Then, we pick any covariance function $\kappa(\mathbf{x}, \mathbf{x}')$ to construct the covariance matrix $\mathbf{K}$ like equation (4.87). We use the Cholesky factorization to obtain the matrix $\mathbf{L}$ such that

$$\mathbf{L}\mathbf{L}^T = \mathbf{K} \tag{4.88}$$

Next, we generate N independently identically distributed data points $\mathbf{v} = [v_1, v_2, \ldots, v_N]^T$ from a zero mean, unit variance Gaussian distribution such that

$$\mathbf{E}[\mathbf{v}] = [0, 0, \ldots, 0]^T \tag{4.89}$$

$$\mathbf{E}\left[\mathbf{v}\mathbf{v}^T\right] = \mathbf{I} \tag{4.90}$$

where $\mathbf{I}$ is the N-by-N identity matrix. Therefore, the vector $\mathbf{y} = \mathbf{L}\mathbf{v}$ is one possible realization of the Gaussian process sampled at $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, because

$$\begin{aligned}\mathbf{E}[\mathbf{y}] &= \mathbf{L} \times \mathbf{E}[\mathbf{v}] = 0 \\ \mathbf{E}\left[\mathbf{y}\mathbf{y}^T\right] &= \mathbf{E}\left[\mathbf{L}\mathbf{v}(\mathbf{L}\mathbf{v})^T\right] \\ &= \mathbf{L} \times \mathbf{E}\left[\mathbf{v}\mathbf{v}^T\right] \times \mathbf{L}^T \\ &= \mathbf{L} \times \mathbf{I} \times \mathbf{L}^T = \mathbf{K}\end{aligned} \tag{4.91}$$

In this experiment, we set $N = 100$. $\mathbf{x}$ is real and one dimensional. $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ are uniformly sampled from the interval $[-15, 15]$. We use $\{x_1, x_2, \ldots, x_N\}$ to denote the input. The covariance function is set as

$$\kappa(x_i, x_j) = \exp\left(-\frac{(x_i - x_j)^2}{2}\right) + 0.1\delta_{ij}$$

where δ_{ij} is the Kronecker delta

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \tag{4.92}$$

This component is from the Gaussian noise and 0.1 is the noise variance. A typical realization of the data set is plotted in Figure 4.3.

Now, suppose we only know the functional form of the covariance

$$\kappa(x_i, x_j) = \theta_1 \exp\left(-\frac{(x_i - x_j)^2}{2\theta_2^2}\right) + \theta_3\delta_{ij}$$

and need to estimate the unknown parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3]^T$ from the data. We use the MML method to do the task. In Figure 4.4, we plot the contour

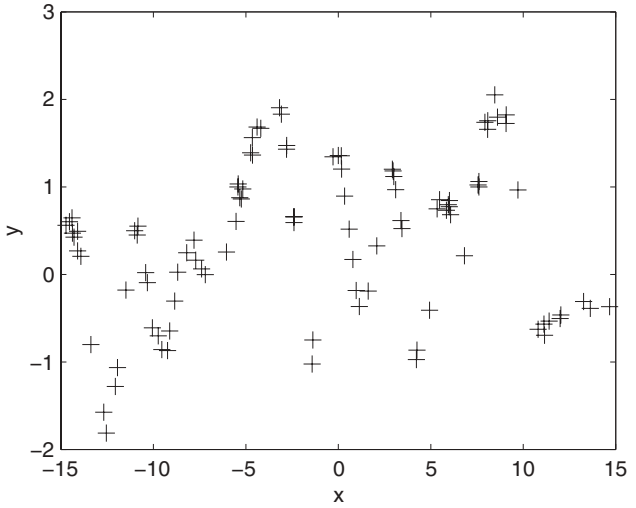


Figure 4.3. Data points sampled from the Gaussian process.

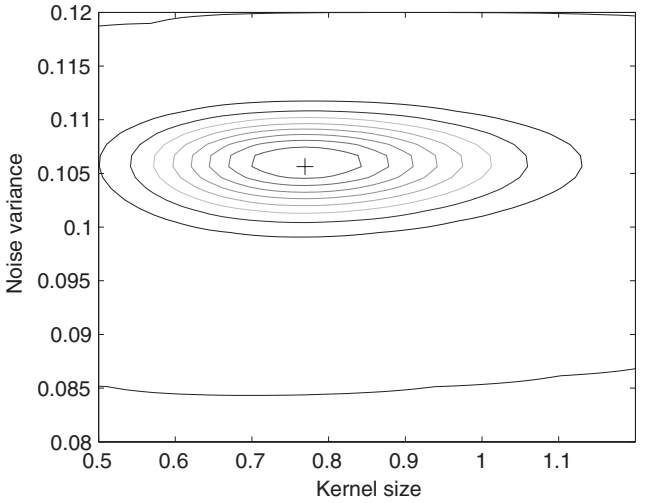


Figure 4.4. Contour of marginal likelihood over the θ_2-θ_3 space with θ_1 fixed at 1.

of the marginal likelihood over the θ_2-θ_3 space with θ_1 fixed at 1. The marker "+" denotes the maximum in the contour. According to this maximum, the noise variance is somewhere around 0.105 and the kernel size is somewhere around 0.73. We know that the estimate of the noise variance is accurate, but the estimate of the kernel size is smaller than the actual value. The reason is that we fix θ_1 when we plot the contour. The maximum marked as "+" in the

Table 4.1. Results of maximum marginal likelihood for hyperparameter estimation in nonlinear regression.

Parameters	Mean	Standard deviation
θ_1	1.0176	0.0827
θ_2	1.0130	0.1637
θ_3	0.0986	0.0103

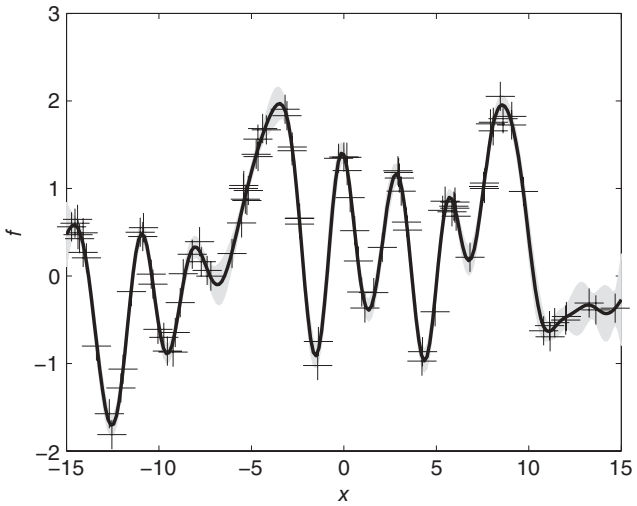


Figure 4.5. The mean and the variance of the posterior distribution of *f*. (The mean is shown as the solid line. The shaded region denotes twice the standard deviation at each input. The original observations are marked as "+").

contour is not the actual maximum if we optimize three parameters at the same time. We will show our results by optimizing three parameters later.

Part 2: We conduct 100 Monte Carlo simulations to test the effectiveness of the methodology of maximum marginal likelihood. In each simulation, we generate different inputs, outputs and initialization to MML. The results are listed in Table 4.1. The results are quite impressive in terms of accuracy.

A typical estimate of the mean and variance of the posterior distribution of f is plotted in Figure 4.5. Notice the large uncertainties at both ends where data are scarce.

4.10 CONCLUSION

The wisdom gained when studying the classic adaptive filtering theory still holds in the kernel space. The convergence rate of KRLS is typically an order of magnitude faster than KLMS. Computational complexity of KLMS scales

linearly with the number of the training data, whereas the computational complexity of KRLS follows a square law.

Another important sparsification scheme, approximate linear dependency test, is introduced and discussed. The ALD approach is suitable for KRLS because it does not require extra computation and is effective in removing redundancy in the data. By combining ALD and NC, we have an enhanced novelty criterion that also checks the prediction error magnitude. This enhanced novelty criterion exhibits superb performance.

The Gaussian process regression theory provides a Bayesian interpretation to linear regression problems, which are closely related to the least-squares approach. As a matter of fact, an online version of GPR is exactly equivalent to KRLS. With the aid of GPR, ALD can be interpreted in terms of prediction uncertainty. Most importantly, we can use GPR to derive a powerful model selection method called maximum marginal likelihood, which is applicable to the kernel selection problem encountered in the design of kernel adaptive filters. As we show in one of the experiments, the maximum marginal likelihood is effective in determining the free parameters in kernel methods.

ENDNOTES

1. **Kernel Recursive Least Squares Algorithm.** Another notable difference in the derivation of Engel et al. [2004] is that the coefficients vector $\mathbf{a}(i)$ is optimized over all the training data, whereas we obtain $\mathbf{a}(i)$ by optimizing over the training data in the dictionary. Denote $\mathbf{\Phi}(i)$ as the matrix of all the transformed input data (including those fail to pass the ALD test), $\mathbf{d}(i)$ as the vector of all the targets, and $\mathbf{\Phi}_c(i)$ as the matrix of the transformed input data, which are in the dictionary $\mathcal{C}(i)$.

   The cost function is defined as

$$J(\mathbf{a}) = \left\| \mathbf{\Phi}(i)^T \mathbf{\Phi}_c(i)\mathbf{a} - \mathbf{d}(i) \right\|^2 \tag{4.93}$$

   Then, an approximation is employed here, stating that

$$\mathbf{\Phi}(i) \simeq \mathbf{\Phi}_c(i)\mathbf{A}(i)^T \tag{4.94}$$

   In other words, all the transformed inputs are approximately represented by linear combinations of the centers in the dictionary. Therefore, the minimizer of equation (4.93) is given by

$$\mathbf{a}(i) = \mathbf{G}_c(i)^{-1}\mathbf{A}(i)^{\dagger}\mathbf{d}(i)$$

   with $\mathbf{A}(i)^{\dagger}$ denoting the pseudoinverse of $\mathbf{A}(i)$ and $\mathbf{G}_c(i)$ denoting the Gram matrix of the centers. At the next iteration, i.e., $i + 1$, there are two possible cases:

   Case 1: $\varphi(\mathbf{u}(i + 1))$ is ALD on $\mathcal{C}(i)$, i.e.,

$$\text{dis}_2 = \min_{\forall \mathbf{b}} \left\| \boldsymbol{\varphi}(\mathbf{u}(i+1)) - \sum_{\mathbf{c}_j \in \mathcal{C}(i)} \mathbf{b}_j \boldsymbol{\varphi}(\mathbf{c}_j) \right\| < \delta_3$$

In this case, the dictionary is unchanged, so we have $\mathcal{C}(i+1) = \mathcal{C}(i)$, $\Phi_c(i+1) = \Phi_c(i)$, and $\mathbf{G}_c(i+1) = \mathbf{G}_c(i)$. Only $\mathbf{A}$ changes as follows:

$$\mathbf{A}(i+1) = \left[\mathbf{A}(i)^T, \mathbf{b}(i+1)\right]^T$$

where $\mathbf{b}(i+1)$ is the linear coefficients achieving the minimum distance dis_2, i.e.,

$$\mathbf{b}(i+1) = \mathbf{G}_c(i)^{-1}\mathbf{h}(i+1) \tag{4.95}$$

where

$$\mathbf{h}(i+1) = [\kappa(\mathbf{u}(i+1), \mathbf{c}_1), \cdots, \kappa(\mathbf{u}(i+1), \mathbf{c}_{m_i})]^T$$

Therefore

$$\begin{aligned} \mathbf{A}(i+1)^T\mathbf{A}(i+1) &= \mathbf{A}(i)^T\mathbf{A}(i) + \mathbf{b}(i+1)\mathbf{b}(i+1)^T \\ \mathbf{A}(i+1)^T\mathbf{d}(i+1) &= \mathbf{A}(i)^T\mathbf{d}(i) + \mathbf{b}(i+1)d(i+1) \end{aligned} \tag{4.96}$$

Defining $\mathbf{P}(i+1) = (\mathbf{A}(i+1)^T\mathbf{A}(i+1))^{-1}$ and applying the matrix inversion lemma, we have the recursive formula:

$$\mathbf{P}(i+1) = \mathbf{P}(i) - \frac{\mathbf{P}(i)\mathbf{b}(i+1)\mathbf{b}(i+1)^T\mathbf{P}(i)}{1+\mathbf{b}(i+1)^T\mathbf{P}(i)\mathbf{b}(i+1)} \tag{4.97}$$

By introducing

$$\mathbf{q}(i+1) = \frac{\mathbf{P}(i)\mathbf{b}(i+1)}{1+\mathbf{b}(i+1)^T\mathbf{P}(i)\mathbf{b}(i+1)}$$

we can derive the recursive formula for $\mathbf{a}(i+1)$:

$$\begin{aligned} \mathbf{a}(i+1) &= \mathbf{G}_c(i+1)^{-1}\mathbf{A}(i+1)^{\dagger}\mathbf{d}(i+1) \\ &= \mathbf{G}_c(i+1)^{-1}P(i+1)A(i+1)^T\mathbf{d}(i+1) \\ &= \mathbf{G}_c(i+1)^{-1}\left(\mathbf{P}(i) - \mathbf{q}(i+1)\mathbf{b}(i+1)^T\mathbf{P}(i)\right)\left(\mathbf{A}(i)^T\mathbf{d}(i) + \mathbf{b}(i+1)d(i+1)\right) \\ &= \underbrace{\mathbf{G}_c(i)^{-1}\mathbf{P}(i)\mathbf{A}(i)^T\mathbf{d}(i)}_{\text{term 1}} + \mathbf{G}_c(i)^{-1}\Bigg[\mathbf{P}(i)\mathbf{b}(i+1)d(i+1) \\ &\quad - \underbrace{\mathbf{q}(i+1)\mathbf{b}(i+1)^T\mathbf{P}(i)\mathbf{A}(i)^T\mathbf{d}(i)}_{\text{term 2}} - \underbrace{\mathbf{q}(i+1)\mathbf{b}(i+1)^T\mathbf{P}(i)\mathbf{b}(i+1)d(i+1)}_{\text{term 3}}\Bigg] \end{aligned} \tag{4.98}$$

Notice that in the above equation, we have

$$\begin{aligned} \underbrace{\mathbf{G}_c(i)^{-1}\mathbf{P}(i)\mathbf{A}(i)^T\mathbf{d}(i)}_{\text{term 1}} &= \mathbf{G}_c(i)^{-1}\left(\mathbf{A}(i)^T\mathbf{A}(i)\right)^{-1}\mathbf{A}(i)^T\mathbf{d}(i) \\ &= \mathbf{G}_c(i)^{-1}\mathbf{A}(i)^{\dagger}\mathbf{d}(i) \\ &= \mathbf{a}(i) \end{aligned} \tag{4.99}$$

$$\underbrace{\mathbf{q}(i+1)\mathbf{b}(i+1)^T\mathbf{P}(i)\mathbf{A}(i)^T\mathbf{d}(i)}_{\text{term 2}} = \mathbf{q}(i+1)\mathbf{h}(i+1)^T\mathbf{G}_c(i)^{-1}\mathbf{P}(i)\mathbf{A}(i)^T\mathbf{d}(i)$$
$$= \mathbf{q}(i+1)\mathbf{h}(i+1)^T\mathbf{a}(i) \tag{4.100}$$

$$\underbrace{\mathbf{q}(i+1)\mathbf{b}(i+1)^T\mathbf{P}(i)\mathbf{b}(i+1)d(i+1)}_{\text{term 3}}$$
$$= \mathbf{q}(i+1)\left[1+\mathbf{b}(i+1)^T\mathbf{P}(i)\mathbf{b}(i+1)-1\right]d(i+1)$$
$$= \mathbf{q}(i+1)\left[1+\mathbf{b}(i+1)^T\mathbf{P}(i)\mathbf{b}(i+1)\right]d(i+1)-\mathbf{q}(i+1)d(i+1)$$
$$= \mathbf{P}(i)\mathbf{b}(i+1)d(i+1)-\mathbf{q}(i+1)d(i+1) \tag{4.101}$$

Substituting these results into equation (4.98) yields

$$\mathbf{a}(i+1) = \mathbf{a}(i)+\mathbf{G}_c(i)^{-1}\mathbf{q}(i+1)\left[d(i+1)-\mathbf{h}(i+1)^T\mathbf{a}(i)\right]$$
$$= \mathbf{a}(i)+\mathbf{G}_c(i)^{-1}\mathbf{q}(i+1)e(i+1) \tag{4.102}$$

where

$$e(i+1) = d(i+1)-\mathbf{h}(i+1)^T\mathbf{a}(i)$$

is the prediction error. Notice that the dimensionality of $\mathbf{a}(i+1)$ is the same as that of $\mathbf{a}(i)$, and the system needs to store and update the matrix $\mathbf{P}(i+1)$ in addition to $\mathbf{G}_c(i+1)$.

Case 2: $\text{dis}_2 > \delta_3$. $\mathbf{u}(i+1)$ is added into the dictionary. $\mathbf{G}_c(i+1)$ is updated in the same way as in equation (4.23). Because $\varphi(\mathbf{u}(i+1))$ is exactly represented by itself, we have

$$\mathbf{A}(i+1) = \begin{bmatrix}\mathbf{A}(i) & \mathbf{0}\\ \mathbf{0}^T & 1\end{bmatrix}$$
$$\mathbf{P}(i+1) = \begin{bmatrix}\mathbf{P}(i) & \mathbf{0}\\ \mathbf{0}^T & 1\end{bmatrix} \tag{4.103}$$

Therefore, the update rule for $\mathbf{a}(i+1)$ is

$$\mathbf{a}(i+1) = \mathbf{G}_c(i+1)^{-1}\mathbf{P}(i+1)\mathbf{A}(i+1)^T\mathbf{d}(i+1)$$
$$= \mathbf{G}_c(i+1)^{-1}\begin{bmatrix}\mathbf{P}(i)\mathbf{A}(i)\mathbf{d}(i)\\ \mathbf{d}(i+1)\end{bmatrix}$$
$$= r(i+1)^{-1}\begin{bmatrix}\mathbf{G}_c(i)^{-1}r(i+1)+\mathbf{b}(i+1)\mathbf{b}(i+1)^T & -\mathbf{b}(i+1)\\ -\mathbf{b}(i+1)^T & 1\end{bmatrix}\begin{bmatrix}\mathbf{P}(i)\mathbf{A}(i)\mathbf{d}(i)\\ d(i+1)\end{bmatrix}$$
$$= \begin{bmatrix}\mathbf{a}(i)-\mathbf{b}(i+1)r(i+1)^{-1}e(i+1)\\ r(i+1)^{-1}e(i+1)\end{bmatrix} \tag{4.104}$$

where

$$r(i+1) = \kappa(\mathbf{u}(i+1),\mathbf{u}(i+1))-\mathbf{h}(i+1)^T\mathbf{G}_c(i)^{-1}\mathbf{h}(i+1) = \text{dis}_2$$

This update rule is the same as in equation (4.26), although we start from the different optimization cost functions. Notice that the update rule does not depend on the matrix $\mathbf{P}(i)$.

2. **Gaussian Process Regression.** The use of Gaussian process prediction in time-series analysis goes back at least as far as the work of Kolmogorov [1941] and Wiener [1949]. In the geostatistics field, Gaussian process regression is known as *kriging* [Matheron, 1973, Journel and Huijbregts, 1978]. Thompson [1956] also used it in meteorology, and Whittle [1963] used it in spatial prediction. Later, O'Hagan [1978] presented the general GPR theory with an arbitrary form of covariance function. Sacks et al. [1989] and his coworkers examined its use in the design of computer experiments. Williams and Rasmussen [1996] described GPR in a machine learning context and presented the maximum marginal likelihood method to optimize the parameters in the covariance function.

   An excellent tutorial book on this topic is Rasmussen and Williams [2006].

5

EXTENDED KERNEL RECURSIVE LEAST-SQUARES ALGORITHM

In this chapter, the kernel recursive least-squares algorithm (KRLS) will be used to implement state space models in reproducing kernel Hilbert spaces (RKHS). Nonlinear state space models are useful in their own right and will open a new research direction in the area of nonlinear Kalman filtering on par with the extended Kalman filter, the cubature Kalman filter, and the particle filter.[1] The first step in this exciting road is to derive an extended recursive least squares algorithm (EX-RLS) in RKHS, and this will be the aim of this chapter.

The chapter will address first the difficulty of deriving an extended kernel recursive least squares algorithm (EX-KRLS) directly from EX-RLS and KRLS. Two important theorems are provided to overcome the difficulty and the simplest of the EX-KRLS algorithms (a scalar state model) will be derived. Exponentially weighted KRLS (EW-KRLS) and random-walk KRLS are also discussed briefly as special cases of this model. Then, we address the issue of creating an arbitrary state-space model using a finite-rank assumption. This algorithm requires new ways to estimate the state model that have not been fully developed yet, so it is just the first step to practical nonlinear state models in RKHS [Liu and Príncipe, 2008a, Liu et al., 2009].

We also address the important issue of how general is this approach to create an arbitrary nonlinear state space model in the input space. We present a theorem that indicates that in principle a linear state model in RKHS is indeed equivalent to an arbitrary state model in the input space, but there are

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

still some limiting assumptions in terms of the noise model that need future research.

We present experimental results showing that EX-KRLS is superior to alternative models (including particle filters) to track a nonlinear time-varying communication channel. We also show that EX-KRLS also outperforms other nonlinear models to identify the Lorenz system, which is a well-known nonlinear dynamical system.

5.1 EXTENDED RECURSIVE LEAST-SQUARES ALGORITHM

The shortcoming of RLS (and KRLS) is that it has a poor tracking performance. From the viewpoint of state-space model, RLS implicitly assumes that the data satisfy [Haykin, 2002]

$$\begin{aligned} \mathbf{x}(i+1) &= \mathbf{x}(i) \\ d(i) &= \mathbf{u}(i)^T \mathbf{x}(i) + v(i) \end{aligned} \tag{5.1}$$

that is, the state $\mathbf{x}(i)$ is fixed over time, and therefore, the optimal estimate of the state $\mathbf{w}(i)$ cannot track variations. To improve tracking ability, several techniques can be employed. For example, the data can be exponentially weighted over time or a truncated window can be applied on the training data (which also can be viewed as a special type of weighting).

As is known from sequential estimation, a more general linear state-space model is

$$\begin{aligned} \mathbf{x}(i+1) &= \mathbf{A}\mathbf{x}(i) + \mathbf{n}(i) && \text{(state model)} \\ d(i) &= \mathbf{u}(i)^T \mathbf{x}(i) + v(i) && \text{(observation model)} \end{aligned} \tag{5.2}$$

with $\mathbf{A}$ as the state transition matrix, $\mathbf{n}(i)$ the state noise, and $v(i)$ the observation noise. The *state model* is also called the *process equation* or *process model*, and the *observation model* is also known as *measurement equation* in the literature. We assume both noises are Gaussian distributed, white spatially and temporally for simplicity, i.e.,

$$\begin{aligned} \mathbf{E}\left[\mathbf{n}(i)\mathbf{n}(j)^T\right] &= \begin{cases} q_1\mathbf{I}, & i = j \\ \mathbf{0}, & i \neq j \end{cases} \\ \mathbf{E}[v(i)v(j)] &= \begin{cases} q_2, & i = j \\ 0, & i \neq j \end{cases} \end{aligned} \tag{5.3}$$

The state model describes an unknown physical stochastic phenomenon denoted by the L-by-1 state vector $\mathbf{x}(i)$ as the output of a linear dynamical

system excited by the white noise $\mathbf{n}(i)$. The observation model relates the observable output of the system $d(i)$ to the state $\mathbf{x}(i)$. A two-step sequential estimation algorithm to update the state estimate was proposed in Kalman [1960]. It is assumed that $\mathbf{x}(1)$, which is the initial value of the state, is uncorrelated with both $\mathbf{n}(i)$ and $v(i)$ for $i \geq 1$. Two noise processes $\mathbf{n}(i)$ and $v(i)$ are statistically independent. Parameters $\mathbf{A}$, q_1 and q_2 are known a priori.

We use $\mathbf{w}(i-1)$ to denote the optimal estimate of $\mathbf{x}(i)$ given the observations starting at time $j = 1$ and extending up to and including time $i-1$, i.e., $\{\mathbf{u}(1), d(1)\}, \{\mathbf{u}(2), d(2)\}, \ldots, \{\mathbf{u}(i-1), d(i-1)\}$, and $\mathbf{w}(i)$ to denote the optimal estimate of $\mathbf{x}(\mathrm{i}+1)$ given the observations starting at time $j = 1$ and extending up to and including time i. As in RLS, we define the *innovations process* associated with $d(i)$ as

$$e(i) = d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i) \tag{5.4}$$

$e(i)$ is the prediction error for the observed $d(i)$ and also represents the new information in $d(i)$. The variance of the innovations process $e(i)$ is defined by

$$\begin{aligned} r(i) &= \mathbf{E}\left[e(i)^2\right] \\ &= \mathbf{E}\left[\left(d(i) - \mathbf{w}^T(i-1)\mathbf{u}(i)\right)^2\right] \\ &= \mathbf{E}\left[\left(\mathbf{x}^T(i-1)\mathbf{u}(i) + v(i) - \mathbf{w}^T(i-1)\mathbf{u}(i)\right)^2\right] \\ &= \mathbf{E}\left[\left((x(i-1) - \mathbf{w}(i-1))^T\mathbf{u}(i) + v(i)\right)^2\right] \\ &= \mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i) + q_2 \end{aligned} \tag{5.5}$$

where $\mathbf{P}(i-1)$ is the *state-error correlation matrix*

$$\mathbf{P}(i-1) = \mathbf{E}\left[(\mathbf{x}(i-1) - \mathbf{w}(i-1))(\mathbf{x}(i-1) - \mathbf{w}(i-1))^T\right] \tag{5.6}$$

To proceed, we need to introduce an important concept, called *Kalman gain* or simply gain vector here, and a fundamental relation between the current optimal estimate and the previous optimal estimate

$$\mathbf{w}(i) = \mathbf{A}\mathbf{w}(i-1) + \mathbf{k}(i)e(i) \tag{5.7}$$

which shows that we can compute the minimum mean-square estimate $\mathbf{w}(i)$ of the state of a linear dynamical system by adding to the previous estimate $\mathbf{w}(i-1)$, which is premultiplied by the transition matrix $\mathbf{A}$, a correction term equal to $\mathbf{k}(i)e(i)$. The correction term equals the prediction error $e(i)$ premultiplied by the gain vector $\mathbf{k}(i)$. This equation is similar to the one in RLS, except the step of premultiplying the transition matrix $\mathbf{A}$. Furthermore, by the

theory of optimal least-squares estimation, we can express the gain vector $\mathbf{k}(i)$ as

$$\mathbf{k}(i) = \mathbf{A}\mathbf{P}(i-1)\mathbf{u}(i)/r(i) \tag{5.8}$$

There now remains only the problem of finding a recursive way of computing the predicted state-error correlation matrix $\mathbf{P}(i-1)$. By using the famous *Riccati equation*, we have

$$\mathbf{P}(i) = \mathbf{A}\left[\mathbf{P}(i-1) - \mathbf{A}^{-1}\mathbf{k}(i)\mathbf{u}(i)^T\mathbf{P}(i-1)\right]\mathbf{A}^T + q_1\mathbf{I} \tag{5.9}$$

Therefore, by initializing $\mathbf{w}(0) = 0$ and $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$, we can compute $r(1)$, $\mathbf{k}(1)$ and then update $\mathbf{w}(1)$, $\mathbf{P}(1)$ with $\{\mathbf{u}(1), d(1)\}$ supplied. The recursion can go on as long as the observations are available.

The extended RLS recursion is summarized in Algorithm 9.

Algorithm 9. The extended recursive least squares algorithm

Initialization

$\mathbf{w}(0) = 0$, $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$

Computation

Iterate for $i \geq 1$

$$\begin{aligned}
r(i) &= q_2 + \mathbf{u}(i)^T\mathbf{P}(i-1)\mathbf{u}(i) \\
\mathbf{k}(i) &= \mathbf{A}\mathbf{P}(i-1)\mathbf{u}(i)/r(i) \\
e(i) &= d(i) - \mathbf{u}(i)^T\mathbf{w}(i-1) \\
\mathbf{w}(i) &= \mathbf{A}\mathbf{w}(i-1) + \mathbf{k}(i)e(i) \\
\mathbf{P}(i) &= \mathbf{A}\mathbf{P}(i-1)\mathbf{A}^T - \mathbf{k}(i)\mathbf{k}(i)^T r(i) + q_1\mathbf{I}
\end{aligned} \tag{5.10}$$

The above derivation is based on solving a stochastic problem. It has an equivalent deterministic formulation. Indeed, the solution of this state-space estimation problem amounts to solving the following least-squares cost function[2]:

$$\min_{\{\mathbf{x}(1),\mathbf{n}(1),\cdots,\mathbf{n}(i)\}}\left[q_2^{-1}\sum_{j=1}^{i}\left|d(j) - \mathbf{u}(j)^T\mathbf{x}(j)\right|^2 + \lambda\|\mathbf{x}(1)\|^2 + q_1^{-1}\sum_{j=1}^{i}\|\mathbf{n}(j)\|^2\right],$$
$$\text{subject to} \quad \mathbf{x}(j+1) = \mathbf{A}\mathbf{x}(j) + \mathbf{n}(j) \tag{5.11}$$

where λ is the regularization parameter to control the initial state-vector norm and q_1, q_2 are the noise variances. In this special case, equation (5.11) is equivalent to

$$\min_{\{\mathbf{x}(1),\mathbf{n}(1),\cdots,\mathbf{n}(i)\}}\left[\sum_{j=1}^{i}\left|d(j) - \mathbf{u}(j)^T\mathbf{x}(j)\right|^2 + \lambda\|\mathbf{x}(1)\|^2 + q^{-1}\sum_{j=1}^{i}\|\mathbf{n}(j)\|^2\right],$$
$$\text{subject to} \quad \mathbf{x}(j+1) = \mathbf{A}\mathbf{x}(j) + \mathbf{n}(j) \tag{5.12}$$

where $q = q_1/q_2$, which indicates the trade-off between the measurement error and state noise. Therefore, the recursion in Algorithm 9 is equivalent to

$$\begin{aligned} r(i) &= 1 + \mathbf{u}(i)^T \mathbf{P}(i-1)\mathbf{u}(i) \\ \mathbf{k}(i) &= \mathbf{A}\mathbf{P}(i-1)\mathbf{u}(i)/r(i) \\ e(i) &= d(i) - \mathbf{u}(i)^T \mathbf{w}(i-1) \\ \mathbf{w}(i) &= \mathbf{A}\mathbf{w}(i-1) + \mathbf{k}(i)e(i) \\ \mathbf{P}(i) &= \mathbf{A}\mathbf{P}(i-1)\mathbf{A}^T - \mathbf{k}(i)\mathbf{k}(i)^T r(i) + q\mathbf{I} \end{aligned} \tag{5.13}$$

5.2 EXPONENTIALLY WEIGHTED EXTENDED RECURSIVE LEAST-SQUARES ALGORITHM

Similar to RLS, we may introduce a forgetting factor to weight more on recent data. The exponentially weighted cost function is

$$\min_{\{\mathbf{x}(1),\mathbf{n}(1),\cdots,\mathbf{n}(i)\}} \left[\sum_{j=1}^{i} \beta^{i-j} \left| d(j) - \mathbf{u}(j)^T \mathbf{x}(j) \right|^2 + \lambda \beta^i \|\mathbf{x}(1)\|^2 + q^{-1} \sum_{j=1}^{i} \beta^{i-j} \|\mathbf{n}(j)\|^2 \right],$$
$$\text{subject to} \quad \mathbf{x}(j+1) = \mathbf{A}\mathbf{x}(j) + \mathbf{n}(j) \tag{5.14}$$

where β is the forgetting factor on the past data, λ is the regularization parameter to control the initial state-vector norm, and q provides a trade-off between the modeling variation and measurement disturbance. Repeating the same argument in the previous section, we have the exponentially weighted extended RLS Algorithm 10.

Algorithm 10. The exponentially weighted extended recursive least-squares algorithm

Initialization

$\mathbf{w}(0) = 0$, $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$

Computation

Iterate for $i \geq 1$

$$\begin{aligned} r(i) &= \beta^i + \mathbf{u}(i)^T \mathbf{P}(i-1)\mathbf{u}(i) \\ \mathbf{k}(i) &= \mathbf{A}\mathbf{P}(i-1)\mathbf{u}(i)/r(i) \\ e(i) &= d(i) - \mathbf{u}(i)^T \mathbf{w}(i-1) \\ \mathbf{w}(i) &= \mathbf{A}\mathbf{w}(i-1) + \mathbf{k}(i)e(i) \\ \mathbf{P}(i) &= \mathbf{A}\mathbf{P}(i-1)\mathbf{A}^T - \mathbf{k}(i)\mathbf{k}(i)^T r(i) + \beta^i q\mathbf{I} \end{aligned} \tag{5.15}$$

It is clear that EX-RLS is a special case of EW-EX-RLS when $\beta = 1$, and we will use EX-RLS to refer to the more general exponentially weighted algorithm. This algorithm is the basis to derive similar algorithms in RKHS.

5.3 EXTENDED KERNEL RECURSIVE LEAST-SQUARES ALGORITHM

Starting from this section, we aim to derive a similar algorithm in general RKHS. In the feature space $\mathbb{F}$, the model becomes

$$\begin{aligned} \mathbf{x}(i+1) &= \mathbf{A}\mathbf{x}(i) + \mathbf{n}(i) \\ d(i) &= \boldsymbol{\varphi}(i)^T \mathbf{x}(i) + v(i) \end{aligned} \tag{5.16}$$

which is similar to equation (5.2) except that the input is $\boldsymbol{\varphi}(i) \in \mathbb{F}$ instead of $\mathbf{u}(i) \in \mathbb{U}$. Notice that $\mathbf{x}(i) \in \mathbb{F}$ now lies in a possibly infinite dimensional feature space, and $\mathbf{A} : \mathbb{F} \to \mathbb{F}$ becomes an operator, rigorously speaking.

The first question is: How expressive can this model be? The following theorem shows that a linear state model in RKHS [equation (5.16)] is equivalent to an arbitrary nonlinear and noiseless model in the input space.

Theorem 5.1. *Assume a general nonlinear state-space model*

$$\begin{aligned} \mathbf{s}(i+1) &= g(\mathbf{s}(i)) \\ d(i) &= h(\mathbf{u}(i), \mathbf{s}(i)) + v(i) \end{aligned} \tag{5.17}$$

where $\mathbf{s} \in \mathbb{S}$ *is the original state vector and* $g : \mathbb{S} \to \mathbb{S}$ *and* $h : \mathbb{U} \times \mathbb{S} \to \mathbb{S}$ *are general continuous nonlinear functions. There exists a transformed input vector* $\boldsymbol{\varphi}(\mathbf{u})$*, a transformed state vector* $\mathbf{x}(\mathbf{s})$*, and a linear operator* $\mathbf{A}$ *, such that the following equations hold*:

$$\begin{aligned} \mathbf{x}(\mathbf{s}(i+1)) &= \mathbf{A}\mathbf{x}(\mathbf{s}(i)) \\ d(i) &= \boldsymbol{\varphi}(\mathbf{u}(i))^T \mathbf{x}(\mathbf{s}(i)) + v(i) \end{aligned} \tag{5.18}$$

and

$$\boldsymbol{\varphi}(\mathbf{u})^T \boldsymbol{\varphi}(\mathbf{u}') = \kappa(\mathbf{u}, \mathbf{u}') \tag{5.19}$$

where κ *is a Mercer kernel with universal approximation capability.*

Proof. It is equivalent to prove that there exist $\boldsymbol{\varphi}(\mathbf{u})$, $\mathbf{x}(\mathbf{s})$ and $\mathbf{A}$ such that

$$\begin{aligned} h(\mathbf{u}, \mathbf{s}) &= \boldsymbol{\varphi}(\mathbf{u})^T \mathbf{x}(\mathbf{s}) \\ \mathbf{x}(g(\mathbf{s})) &= \mathbf{A}\mathbf{x}(\mathbf{s}) \end{aligned} \tag{5.20}$$

Assume the Gaussian kernel $\kappa : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$ is chosen here without loss of generality. To simplify the demonstration, we assume the dimensionality of the induced RKHS is M, which is ∞ in the case of the Gaussian kernel. And the transformed input must be constructed as

$$\boldsymbol{\varphi}(\mathbf{u}) = \left[\sqrt{\varsigma_1}\phi_1(\mathbf{u}), \sqrt{\varsigma_2}\phi_2(\mathbf{u}), \ldots, \sqrt{\varsigma_M}\phi_M(\mathbf{u}), 0, 0, \ldots\right]^T$$

where ς_j and ϕ_j are defined in equation (1.26). 0's are appended to make the total dimension of the transformed vector $2M$.

First of all, by Mercer theorem, we have $\boldsymbol{\varphi}(\mathbf{u})^T\boldsymbol{\varphi}(\mathbf{u}') = \kappa(\mathbf{u}, \mathbf{u}')$. Then, it is proved by Steinwart [2001] that the RKHS induced by the Gaussian kernel has the universal approximation capability, that is, by treating $\mathbf{s}$ as the parameter and $h(\mathbf{u}, \mathbf{s})$ as a function of $\mathbf{u}$, we can represent it as a linear combination of $\left\{\sqrt{\varsigma_j}\,\phi_j(\mathbf{u})\right\}_{j=1}^{M}$, which form a basis of the function space:

$$h(\mathbf{u}, \mathbf{s}) = \sum_{j=1}^{M} \sqrt{\varsigma_j}\,\phi_j(\mathbf{u})\mathbf{x}_j(\mathbf{s})$$

where $\mathbf{x}_j$ is a scalar function of $\mathbf{s}$. Now we can construct the transformed state vector as

$$\mathbf{x}(\mathbf{s}) = [\mathbf{x}_1(\mathbf{s}), \mathbf{x}_2(\mathbf{s}), \ldots, \mathbf{x}_M(\mathbf{s}), \phi_1(\mathbf{s}), \phi_2(\mathbf{s}), \ldots, \phi_M(\mathbf{s})]^T$$

Note that we append 0's at the end of $\boldsymbol{\varphi}(\mathbf{u})$ to match dimension with $\mathbf{x}$ and more importantly

$$\boldsymbol{\varphi}(\mathbf{u})^T\,\mathbf{x}(\mathbf{s}) = h(\mathbf{u}, \mathbf{s})$$

Apparently, $\{\phi_j(\mathbf{s})\}_{j=1}^{M}$ form a basis for another RKHS (with the function domain defined on $\mathbb{S}$) and by the same argument, the composition functions $\mathbf{x}_j \circ g$ and $\phi_j \circ g$ (which are all functions defined from $\mathbb{S}$ to $\mathbb{R}$) can be written as linear combinations of the basis. Putting this fact into a matrix form, we have a $2M \times 2M$ matrix $\mathbf{A}$ (a linear operator) such that

$$\mathbf{x}(g(\mathbf{s})) = \mathbf{A}\mathbf{x}(\mathbf{s})$$

Therefore, the existence is proved by construction. □

This theorem effectively shows that, for any nonlinear state-space model given by equation (5.17), an equivalent representation of a linear state-space model exists in RKHS [equation (5.18)]. This result is of great significance, providing a theoretic foundation to address general nonlinear state-space model estimation problems. Notice that we purposely omit the state noise in equations (5.17) and (5.18), because the noise term makes the proof complicated without providing more insights. In general, the existence of state noise requires linear approximation around operating points, and the approximation error can be modeled as part of $\mathbf{n}(i)$.

Now, the question is how to find a procedure to carry the computation given the model in equation (5.16), noting that many variables exist in high-dimensional spaces. By slightly abusing notation, we view operators as infinite dimensional matrices to treat the recursions in the input space and feature space consistently. Therefore, the recursion to estimate $\mathbf{x}(i)$ in the RKHS is as follows:

Start with $\boldsymbol{\omega}(0) = 0$, $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$, and iterate for $i \geq 1$,

$$\begin{aligned} r(i) &= \beta^i + \boldsymbol{\varphi}(i)^T \mathbf{P}(i-1)\boldsymbol{\varphi}(i) \\ \mathbf{k}(i) &= \mathbf{A}\mathbf{P}(i-1)\boldsymbol{\varphi}(i)/r(i) \\ e(i) &= d(i) - \boldsymbol{\varphi}(i)^T \boldsymbol{\omega}(i-1) \\ \boldsymbol{\omega}(i) &= \mathbf{A}\boldsymbol{\omega}(i-1) + \mathbf{k}(i)e(i) \\ \mathbf{P}(i) &= \mathbf{A}\left[\mathbf{P}(i-1) - \mathbf{P}(i-1)\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T \mathbf{P}(i-1)/r(i)\right]\mathbf{A}^T + \beta^i q\mathbf{I} \end{aligned} \tag{5.21}$$

However, this approach is not feasible in practice because the transformed data $\boldsymbol{\varphi}(i)$ and the state vector $\mathbf{x}(i)$ now lie in a possibly infinite dimensional space $\mathbb{F}$ (consequently the $\mathbf{P}$ matrix could be $\infty \times \infty$ and the vector $\boldsymbol{\omega}$ $\infty \times 1$). We cannot use the matrix inversion lemma as in KRLS because of the complicated constrained least squares cost function [equation (5.11)]. The use of the kernel trick requires a reformulation of the recursion solely in terms of inner product operations between transformed input vectors.

Although it would be most desirable to have such a general state-space model in the RKHS, it turns out to be at best difficult. We focus on two special cases of the state transition operator $\mathbf{A}$ in the next two sections.

5.4 EX-KRLS FOR TRACKING MODELS

For the first special case, we assume the state transition operator has the form $\mathbf{A} = \alpha\mathbf{I}$, then we have the following tracking model:

$$\begin{aligned} \mathbf{x}(i+1) &= \alpha\mathbf{x}(i) + \mathbf{n}(i) \\ d(i) &= \boldsymbol{\varphi}(i)^T \mathbf{x}(i) + v(i) \end{aligned} \tag{5.22}$$

where α is a scaling factor and indicates a uniform change across the vector state, either attenuating or amplifying. If the change is slow over time, then α is close to 1. For example, this model is particularly suitable to track slow fading channels in communications and it provides a theoretical foundation for the exponentially weighted kernel recursive least squares. Furthermore, introducing the state noise is also important, because it provides a framework to study random walk state-space model in RKHS.

Theoretically, the model in equation (5.22) can be used to solve the following problem:

$$\begin{aligned} \mathbf{s}(i+1) &= \tilde{\alpha}\mathbf{s}(i) + \mathbf{n}(i) \\ d(i) &= h(\mathbf{u}(i), \mathbf{s}(i)) + v(i) \end{aligned} \tag{5.23}$$

for any nonlinear function h and with $\tilde{\alpha}$ very close to 1. Because $\tilde{\alpha}$ is close to 1 and the nonlinear mapping between $\mathbf{s}$ and $\mathbf{x}$ is continuous, a single scalar α is sufficient to model this slow change. With this model, the recursion equation (5.21) becomes:

Start with $\boldsymbol{\omega}(0) = 0$, $\mathbf{P}(0) = \lambda^{-1}\mathbf{I}$, and iterate for $i \geq 1$,

$$\begin{aligned} r(i) &= \beta^i + \boldsymbol{\varphi}(i)^T \mathbf{P}(i-1)\boldsymbol{\varphi}(i) \\ \mathbf{k}(i) &= \alpha\mathbf{P}(i-1)\boldsymbol{\varphi}(i)/r(i) \\ e(i) &= d(i) - \boldsymbol{\varphi}(i)^T \boldsymbol{\omega}(i-1) \\ \boldsymbol{\omega}(i) &= \alpha\boldsymbol{\omega}(i-1) + \mathbf{k}(i)e(i) \\ \mathbf{P}(i) &= |\alpha|^2\left[\mathbf{P}(i-1) - \mathbf{P}(i-1)\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T \mathbf{P}(i-1)/r(i)\right] + \beta^i q\mathbf{I} \end{aligned} \tag{5.24}$$

By carefully observing the recursion in equation (5.24), one can conclude that the expression $\boldsymbol{\varphi}(j)^T\mathrm{P}(k)\boldsymbol{\varphi}(i)$ (for any k, i, j), which can be $(1 \times \infty) \times (\infty \times \infty) \times (\infty \times 1)$, plays an important role in the recursion. Making this expression computable in the input space is crucial and is demonstrated by the following theorem.

Theorem 5.2. *The matrices* $\mathbf{P}(j)$ *in equation* (5.24) *are of the following form*:

$$\mathbf{P}(j) = \rho(j)\mathbf{I} - \mathbf{H}(j)\mathbf{Q}(j)\mathbf{H}(j)^T \tag{5.25}$$

where $\rho(j)$ *is a scalar and* $\mathbf{Q}(j)$ *is a* $j \times j$ *real-valued matrix with* $\mathbf{H}(j) = [\boldsymbol{\varphi}(1), \ldots, \boldsymbol{\varphi}(j)]$, *for all* $j > 0$.

Proof. First notice that by equation (5.24)

$$\begin{aligned} \mathbf{P}(0) &= \lambda^{-1}\mathbf{I} \\ r(1) &= \beta + \lambda^{-1}\kappa(\mathbf{u}(1), \mathbf{u}(1)) \\ \mathbf{P}(1) &= \left[\frac{|\alpha|^2}{\lambda} + \beta q\right]\mathbf{I} - \boldsymbol{\varphi}(1)\left[\frac{|\alpha|^2\lambda^{-2}}{\beta + \lambda^{-1}\kappa(\mathbf{u}(1), \mathbf{u}(1))}\right]\boldsymbol{\varphi}(1)^T \end{aligned} \tag{5.26}$$

so the claim is valid for $j = 1$, namely,

$$\rho(1) = |\alpha|^2 \lambda^{-1} + \beta q,$$
$$\mathbf{Q}(1) = \frac{|\alpha|^2 \lambda^{-2}}{\beta + \lambda^{-1}\kappa(\mathbf{u}(1), \mathbf{u}(1))} \tag{5.27}$$

Then, by induction, the proof for all j follows. Assume it is true for $j = i - 1$,

$$\mathbf{P}(i-1) = \rho(i-1)\mathbf{I} - \mathbf{H}(i-1)\mathbf{Q}(i-1)\mathbf{H}(i-1)^T \tag{5.28}$$

By substituting into the last equation of equation (5.24), we have

$$\begin{aligned}\mathbf{P}(i) &= |\alpha|^2\left[\mathbf{P}(i-1) - \mathbf{P}(i-1)\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)r^{-1}(i)\right] + \beta^i q\mathbf{I} \\ &= \left(|\alpha|^2\rho(i-1) + \beta^i q\right)\mathbf{I} \\ &\quad - |\alpha|^2 r^{-1}(i)\mathbf{H}(i)\begin{bmatrix}\mathbf{Q}(i-1)r(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\rho(i-1)\mathbf{z}(i) \\ -\rho(i-1)\mathbf{z}(i)^T & \rho^2(i-1)\end{bmatrix}\mathbf{H}(i)^T\end{aligned} \tag{5.29}$$

where $\mathbf{z}(i) = \mathbf{Q}(i-1)\mathbf{H}(i-1)^T\boldsymbol{\varphi}(i)$. Notice that $r(i)$ is computable by $r(i) = \beta^i + \boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)\boldsymbol{\varphi}(i)$. Therefore,

$$\begin{aligned}\rho(i) &= |\alpha|^2\rho(i-1) + \beta^i q \\ \mathbf{Q}(i) &= \frac{|\alpha|^2}{r(i)}\begin{bmatrix}\mathbf{Q}(i-1)r(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\rho(i-1)\mathbf{z}(i) \\ -\rho(i-1)\mathbf{z}(i)^T & \rho^2(i-1)\end{bmatrix} \\ \mathbf{P}(i) &= \rho(i)\mathbf{I} - \mathbf{H}(i)\mathbf{Q}(i)\mathbf{H}(i)^T\end{aligned} \tag{5.30}$$

□

By Theorem 5.2, the calculation $\boldsymbol{\varphi}(j)^T\mathbf{P}(k)\boldsymbol{\varphi}(i)$ only involves inner product operations between the transformed input vectors, and the kernel trick can be readily applied. Furthermore, we can express the weight vector (which lies in a high-dimensional feature space $\mathbb{F}$) as a linear combination of the previous transformed input vectors as shown in the next theorem.

Theorem 5.3. *The optimal state estimate in equation* (5.24) *is a linear combination of the transformed inputs*

$$\boldsymbol{\omega}(j) = \mathbf{H}(j)\mathbf{a}(j) \tag{5.31}$$

where $\mathbf{a}(j)$ *is a* $j \times 1$ *real-valued vector with* $\mathbf{H}(j) = [\boldsymbol{\varphi}(1), \ldots, \boldsymbol{\varphi}(j)]$, *for all* $j > 0$.

Proof. Notice that by equation (5.24)

$$\boldsymbol{\omega}(0) = 0,$$
$$\boldsymbol{\omega}(1) = \frac{\alpha\lambda^{-1}d(1)\boldsymbol{\varphi}(1)}{\beta + \lambda^{-1}\kappa(\mathbf{u}(1), \mathbf{u}(1))} \tag{5.32}$$

so

$$\mathbf{a}(1) = \frac{\alpha\lambda^{-1}d(1)}{\beta + \lambda^{-1}\kappa(\mathbf{u}(1), \mathbf{u}(1))} \tag{5.33}$$

Thus, the claim is valid for $j = 1$. Then, we use induction to prove it is valid for any j. Assume it is true for $i - 1$. By the recursion in equation (5.24) and the result from Theorem 5.2, we have

$$\begin{aligned}
\boldsymbol{\omega}(i) &= \alpha\boldsymbol{\omega}(i-1) + k_p(i)e(i) \\
&= \alpha\mathbf{H}(i-1)\mathbf{a}(i-1) + \alpha\mathbf{P}(i-1)\boldsymbol{\varphi}(i)e(i)/r(i) \\
&= \alpha\mathbf{H}(i-1)\mathbf{a}(i-1) + \alpha\rho(i-1)\boldsymbol{\varphi}(i)e(i)/r(i) - \alpha\mathbf{H}(i-1)\mathbf{z}(i)e(i)/r(i) \\
&= \mathbf{H}(i)\begin{bmatrix} \alpha\mathbf{a}(i-1) - \alpha\mathbf{z}(i)e(i)r^{-1}(i) \\ \alpha\rho(i-1)e(i)r^{-1}(i) \end{bmatrix}
\end{aligned} \tag{5.34}$$

where $\mathbf{z}(i) = \mathbf{Q}(i-1)\mathbf{H}(\mathrm{i}-1)^T\boldsymbol{\varphi}(i)$ as before. This completes the proof. □

Hence, after representing $\boldsymbol{\omega}$ with $\mathbf{a}$, and $\mathbf{P}$ with ρ and $\mathbf{Q}$, we have the following equivalent recursion, which is computable in the input space:

Start with

$$\begin{aligned}
\mathbf{a}(1) &= \frac{\alpha d(1)}{\beta\lambda + \kappa(\mathbf{u}(1), \mathbf{u}(1))} \\
\rho(1) &= |\alpha|^2\lambda^{-1} + \beta q \\
\mathbf{Q}'(1) &= \frac{|\alpha|^2\lambda^{-1}}{\beta\lambda + \kappa(\mathbf{u}(1), \mathbf{u}(1))}
\end{aligned}$$

Iterate for $i > 1$:

$$\begin{aligned}
\mathbf{h}(i) &= \mathbf{H}(i-1)^T\boldsymbol{\varphi}(i) \\
\mathbf{z}'(i) &= \mathbf{Q}'(i-1)\mathbf{h}(i) \\
r(i) &= \beta^i + \rho(i-1)\kappa(\mathbf{u}(i), \mathbf{u}(i)) - \mathbf{h}(i)^T\mathbf{z}'(i) \\
e(i) &= d(i) - \mathbf{h}(i)^T\mathbf{a}(i-1) \\
\mathbf{a}(i) &= \alpha\begin{bmatrix} \mathbf{a}(i-1) - \mathbf{z}'(i)r^{-1}(i)e(i) \\ \rho(i-1)r^{-1}(i)e(i) \end{bmatrix} \\
\rho(i) &= |\alpha|^2\rho(i-1) + \beta^i q \\
\mathbf{Q}'(i) &= \frac{|\alpha|^2}{r(i)}\begin{bmatrix} \mathbf{Q}'(i-1)r(i) + \mathbf{z}'(i)\mathbf{z}'(i)^T & -\rho(i-1)\mathbf{z}'(i) \\ -\rho(i-1)\mathbf{z}'(i)^T & \rho^2(i-1) \end{bmatrix}
\end{aligned}$$

This is sufficient for our purpose, but when λ is small, ρ, $\mathbf{Q}'$, $\mathbf{z}'$, and r are all large, which causes possible numerical issues. And also to understand more

accurately the meanings of these quantities, we do the following change of variables:

$$\mathbf{Q}(i-1) = \mathbf{Q}'(i-1)/\rho(i-1)$$
$$\mathbf{z}(i) = \mathbf{z}'(i)/\rho(i-1)$$
$$r(i) = r(i)/\rho(i-1)$$
$$\rho_r(i) = 1/\rho(i-1)$$

Therefore we have Algorithm 11.

Algorithm 11. The extended kernel recursive least-squares algorithm for the tracking model [equation (5.22)]

Initialization

$$\mathbf{a}(1) = \frac{\alpha d(1)}{\lambda\beta + \kappa(\mathbf{u}(1), \mathbf{u}(1))}$$
$$\rho_r(1) = \lambda\beta \Big/ \left(|\alpha|^2 \beta + \lambda q\right)$$
$$\mathbf{Q}(1) = \frac{|\alpha|^2}{[\beta\lambda + \kappa(\mathbf{u}(1), \mathbf{u}(1))]\left[|\alpha|^2 + \beta\lambda q\right]}$$

Computation

Iterate for $i > 1$:

$$\mathbf{h}(i) = [\kappa(\mathbf{u}(i), \mathbf{u}(1)), \cdots, \kappa(\mathbf{u}(i), \mathbf{u}(i-1))]^T$$
$$\mathbf{z}(i) = \mathbf{Q}(i-1)\mathbf{h}(i)$$
$$r(i) = \beta^i \rho_r(i-1) + \kappa(\mathbf{u}(i), \mathbf{u}(i)) - \mathbf{h}(i)^T \mathbf{z}(i)$$
$$e(i) = d(i) - \mathbf{h}(i)^T \mathbf{a}(i-1)$$
$$\mathbf{a}(i) = \alpha \begin{bmatrix} \mathbf{a}(i-1) - \mathbf{z}(i) r^{-1}(i) e(i) \\ r^{-1}(i) e(i) \end{bmatrix} \tag{5.35}$$
$$\rho_r(i) = \rho_r(i-1) \Big/ \left(|\alpha|^2 + \beta^i q \rho_r(i-1)\right)$$
$$\mathbf{Q}(i) = \frac{|\alpha|^2}{r(i)\left(|\alpha|^2 + \beta^i q \rho_r(i-1)\right)} \begin{bmatrix} \mathbf{Q}(i-1) r(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\mathbf{z}(i) \\ -\mathbf{z}(i)^T & 1 \end{bmatrix}$$

Notice that throughout the iteration, the input vector $\boldsymbol{\varphi}(i)$ only enters in the calculation of $\mathbf{h}(i)$ and $r(i)$, both in the form of inner products. The significance of this reformulation is its independence on the data dimensionality. Compared with equation (5.24), we replaced the recursion on $\boldsymbol{\omega}(i)$ with the one on $\mathbf{a}(i)$, which is $i \times 1$. Furthermore, we replaced the recursion on $\mathbf{P}(i)$ with the ones on $\rho(i)$ and $\mathbf{Q}(i)$, where $\rho(i)$ is a scalar and $\mathbf{Q}(i)$ is $i \times i$. To

summarize, the dimensionality of $\mathbf{a}(i)$, $\rho(i)$, and $\mathbf{Q}(i)$ only depends on the number of training data at time i regardless of the dimensionality of the transformed data $\boldsymbol{\varphi}$.

The learning procedure of EX-KRLS is similar to KRLS as a growing radial-basis-function (RBF) network: It allocates a new unit with $\mathbf{u}(i)$ as the center and $r^{-1}(i)e(i)$ as the coefficient; at the same time, all the previous coefficients are updated by a quantity $-\mathbf{z}(i)r^{-1}(i)e(i)$. Finally, a scalar α is multiplied on $\mathbf{a}(i)$ according to the state update equation. The major difference is the introduction of ρ, which reflects the uncertainty of the state noise. Moreover, we have the following two special cases.

By setting $\alpha = 1$, we have KRLS for the following random walk model:

$$\begin{aligned} \mathbf{x}(i+1) &= \mathbf{x}(i) + \mathbf{n}(i) \\ d(i) &= \boldsymbol{\varphi}(i)^T \mathbf{x}(i) + v(i) \end{aligned} \tag{5.36}$$

The recursive equation to estimate the random walk state in RKHS is summarized in Algorithm 12.

Algorithm 12. The random walk kernel recursive least-squares algorithm (RW-KRLS)

Initialization

$$\mathbf{a}(1) = \frac{d(1)}{\lambda\beta + \kappa(\mathbf{u}(1), \mathbf{u}(1))}$$

$$\rho_r(1) = \lambda\beta/(\beta + \lambda q)$$

$$\mathbf{Q}(1) = \frac{1}{(\beta\lambda + \kappa(\mathbf{u}(1), \mathbf{u}(1)))(1 + \beta\lambda q)}$$

Computation

Iterate for $i > 1$:

$$\mathbf{h}(i) = [\kappa(\mathbf{u}(i), \mathbf{u}(1)), \cdots, \kappa(\mathbf{u}(i), \mathbf{u}(i-1))]^T$$

$$\mathbf{z}(i) = \mathbf{Q}(i-1)\mathbf{h}(i)$$

$$r(i) = \beta^i \rho_r(i-1) + \kappa(\mathbf{u}(i), \mathbf{u}(i)) - \mathbf{h}(i)^T \mathbf{z}(i)$$

$$e(i) = d(i) - \mathbf{h}(i)^T \mathbf{a}(i-1)$$

$$\mathbf{a}(i) = \begin{bmatrix} \mathbf{a}(i-1) - \mathbf{z}(i)r^{-1}(i)e(i) \\ r^{-1}(i)e(i) \end{bmatrix}$$

$$\rho_r(i) = \rho_r(i-1)/\left(1 + \beta^i q \rho_r(i-1)\right)$$

$$\mathbf{Q}(i) = \frac{1}{r(i)\left(1 + \beta^i q \rho_r(i-1)\right)} \begin{bmatrix} \mathbf{Q}(i-1)r(i) + \mathbf{z}(i)\mathbf{z}(i)^T & -\mathbf{z}(i) \\ -\mathbf{z}(i)^T & 1 \end{bmatrix}$$

By setting $q = 0$ and $\alpha = 1$, we have the exponentially weighted KRLS as summarized in Algorithm 8. Note that $\rho(i)$ becomes a constant and is no longer needed. Setting $\beta = 1$, we have KRLS back.

To summarize, all the algorithms discussed in this section can be regarded as nontrivial extensions of KRLS, with an explicit state transition process. They are suitable to model nonlinear systems with a slow fading and a small variation in state, improving the tracking ability of KRLS.

The implementation of these algorithms is straightforward. We need to store $\mathbf{a}(i)$, $\rho(i)$, and $\mathbf{Q}(i)$, so the total time and space complexities are both $O(i^2)$ at iteration i. The network size increases linearly with the number of training data as in kernel least-mean-square algorithm (KLMS) and kernel affine projection algorithm (KAPA). The novelty criterion can be used readily in the EX-KRLS. Here, we discuss how to use approximate linear dependency(ALD) in EX-KRLS. Note that $r(i)$ in the EX-KRLS equation (5.35) plays a similar role to $r(i)$ in KRLS. Although its meaning is not as clear as in KRLS, it is at least a good approximation of the distance especially when α, β are close to 1 and q is small, which is usually valid in slow-fading applications. Therefore, ALD is readily applicable for EX-KRLS, EW-KRLS, and RW-KRLS without extra computation. Of course, because of the long-term effect of state transition model, EX-KRLS can deviate significantly from KRLS when the iteration goes on for a long time, so some caution is needed here. The worst case is to compute dis_2 at each iteration, which also scales with $O(m_i^2)$.

5.5 EX-KRLS WITH FINITE RANK ASSUMPTION

As one can expect from the previous derivation, it is difficult to have the recursion computable with a general state transition operator $\mathbf{A}$. Here, we propose a subspace technique to make the recursion feasible. In this section, the state transition operator is assumed to be

$$\mathbf{A} = \sum_{m=1}^{M} \sum_{j=1}^{J} b_{m,j} \boldsymbol{\varphi}(\mathbf{c}_m) \boldsymbol{\varphi}(\mathbf{e}_j)^T \tag{5.37}$$

The mathematical meaning of this assumption can be interpreted as follows: Define $\text{nul}(\mathbf{A}) = \{x \in \mathbb{F} : \mathbf{A}x = 0\}$ as the null space of $\mathbf{A}$ and $\text{im}(\mathbf{A}) = \{y \in \mathbb{F} : y = \mathbf{A}x, x \in \mathbb{F}\}$ the image or range of $\mathbf{A}$. Then, $\{\boldsymbol{\varphi}(\mathbf{c}_m)\}_{m=1}^{M}$ is a set of basis of $\text{im}(\mathbf{A})$, and $\{\boldsymbol{\varphi}(\mathbf{e}_j)\}_{j=1}^{J}$ is a set of basis of the orthogonal complement of $\text{nul}(\mathbf{A})$. The dimensionality of the two subspaces $\text{im}(\mathbf{A})$ and $\text{nul}(\mathbf{A})^{\perp}$ are assumed to be finite, i.e., $M < \infty$ and $J < \infty$. Actually, by the rank-nullity theorem [Williams, 2007], $M = J$. And M is also called the rank of $\mathbf{A}$, thus, the name of the assumption. $b_{m,j}$ is a set of real-valued coefficients.

For any state vector $\mathbf{x}$, the operator $\mathbf{A}$ transforms $\mathbf{x}$ as a linear combination of $\{\boldsymbol{\varphi}(\mathbf{c}_m)\}_{m=1}^{M}$ with $\sum_{j=1}^{J} b_{m,j} \boldsymbol{\varphi}(\mathbf{e}_j)^T \mathbf{x}$ as the coefficients:

$$\mathbf{Ax} = \sum_{m=1}^{M} \boldsymbol{\varphi}(\mathbf{c}_m)\left(\sum_{j=1}^{J} b_{m,j}\boldsymbol{\varphi}(\mathbf{e}_j)^T\mathbf{x}\right) \tag{5.38}$$

Denoting $\mathbf{C} = [\boldsymbol{\varphi}(\mathbf{c}_1), \ldots, \boldsymbol{\varphi}(\mathbf{c}_M)]$, $\mathbf{E} = [\boldsymbol{\varphi}(\mathbf{e}_1), \ldots, \boldsymbol{\varphi}(\mathbf{e}_J)]$, and $\mathbf{B} = [b_{m,j}]_{M\times J}$, we have

$$\mathbf{A} = \mathbf{CBE}^T$$

By this assumption, we can prove the following two theorems like in the previous section.

Theorem 5.4. *By the assumption in equation* (5.37)*, the matrices* $\mathbf{P}(j)$ *in equation* (5.21) *assume the following form:*

$$\mathbf{P}(j) = \rho(j)\mathbf{I} - \mathbf{CQ}(j)\mathbf{C}^T \tag{5.39}$$

where $\rho(j)$ *is a scalar and* $\mathbf{Q}(j)$ *is an* $M \times M$ *real-valued matrix, for all* $j > 0$.

Proof. First notice that by equation (5.21)

$$\begin{aligned}
\mathbf{P}(0) &= \lambda^{-1}\mathbf{I} \\
r(1) &= \beta + \lambda^{-1}\kappa(\mathbf{u}(1), \mathbf{u}(1)) \\
\mathbf{P}(1) &= \mathbf{CBE}^T\left[\lambda^{-1}\mathbf{I} - \frac{\boldsymbol{\varphi}(1)\boldsymbol{\varphi}(1)^T}{\lambda^2 r(1)}\right]\mathbf{EB}^T\mathbf{C}^T + \beta q\mathbf{I} \\
&= C\left[\frac{\mathbf{BE}^T\mathbf{EB}^T}{\lambda} - \frac{\mathbf{BE}^T\boldsymbol{\varphi}(1)\boldsymbol{\varphi}(1)^T\mathbf{EB}^T}{\lambda^2 r(1)}\right]\mathbf{C}^T + \beta q\mathbf{I}
\end{aligned}$$

Denote $\mathbf{G}_\mathbf{E} = \mathbf{E}^T\mathbf{E}$ and $\mathbf{E}^T\boldsymbol{\varphi}(j) = \mathbf{h}_\mathbf{E}(j)$, which are both computable by the kernel trick. So, the claim is valid for $j = 1$, namely,

$$\begin{aligned}
\rho(1) &= \beta q \\
\mathbf{Q}(1) &= -\frac{\mathbf{BG}_\mathbf{E}\mathbf{B}^T}{\lambda} + \frac{\mathbf{Bh}_\mathbf{E}(1)\mathbf{h}_\mathbf{E}(1)^T\mathbf{B}^T}{\lambda^2 r(1)}
\end{aligned}$$

Using mathematical induction, the proof for all j follows. Assume it is true for $j = i - 1$, i.e.,

$$\mathbf{P}(i-1) = \rho(i-1)\mathbf{I} - \mathbf{CQ}(i-1)\mathbf{C}^T \tag{5.40}$$

and $r(i)$ is computable using the kernel trick. By substituting it into the last equation of (5.21), we have

$$\begin{aligned}\mathbf{P}(i) &= \mathbf{C}\mathbf{B}\mathbf{E}^T\left[\mathbf{P}(i-1) - \frac{\mathbf{P}(i-1)\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T\mathbf{P}(i-1)}{r(i)}\right]\mathbf{E}\mathbf{B}^T\mathbf{C}^T + \beta^i q\mathbf{I} \\ &= \beta^i q\mathbf{I} - \mathbf{C}[-\rho(i-1)\mathbf{B}\mathbf{G}_{\mathbf{E}}\mathbf{B}^T - \mathbf{B}\mathbf{G}_{\mathbf{EC}}\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T \\ &\quad + \mathbf{B}\mathbf{h}_{\mathbf{E}}(i)\mathbf{h}_{\mathbf{E}}(i)^T\mathbf{B}^T\rho(i-1)^2/r(i) + \mathbf{B}\mathbf{h}_{\mathbf{E}}(i)\mathbf{h}_{\mathbf{C}}(i)^T\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T\rho(i-1)/r(i) \\ &\quad + \mathbf{B}\mathbf{G}_{\mathbf{EC}}\mathbf{Q}(i-1)\mathbf{h}_{\mathbf{C}}(i)\mathbf{h}_{\mathbf{E}}(i)^T\mathbf{B}^T\rho(i-1)/r(i) \\ &\quad + \mathbf{B}\mathbf{G}_{\mathbf{EC}}\mathbf{Q}(i-1)\mathbf{h}_{\mathbf{C}}(i)\mathbf{h}_{\mathbf{C}}(i)^T\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T/r(i)]\mathbf{C}^T\end{aligned}$$

where $\mathbf{G}_{\mathbf{EC}} = \mathbf{E}^T\mathbf{C}$ and $\mathbf{h}_{\mathbf{C}}(i) = \mathbf{C}^T\boldsymbol{\varphi}(i)$, which are both computable using the kernel trick. Therefore,

$$\begin{aligned}\rho(i) &= \beta^i q \\ \mathbf{Q}(i) &= -\rho(i-1)\mathbf{B}\mathbf{G}_{\mathbf{E}}\mathbf{B}^T - \mathbf{B}\mathbf{G}_{\mathbf{EC}}\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T \\ &\quad + \mathbf{B}\mathbf{h}_{\mathbf{E}}(i)\mathbf{h}_{\mathbf{E}}(i)^T\mathbf{B}^T\rho(i-1)^2/r(i) + \mathbf{B}\mathbf{h}_{\mathbf{E}}(i)\mathbf{h}_{\mathbf{C}}(i)^T\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T\rho(i-1)/r(i) \\ &\quad + \mathbf{B}\mathbf{G}_{\mathbf{EC}}\mathbf{Q}(i-1)\mathbf{h}_{\mathbf{C}}(i)\mathbf{h}_{\mathbf{E}}(i)^T\mathbf{B}^T\rho(i-1)/r(i) \\ &\quad + \mathbf{B}\mathbf{G}_{\mathbf{EC}}\mathbf{Q}(i-1)\mathbf{h}_{\mathbf{C}}(i)\mathbf{h}_{\mathbf{C}}(i)^T\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T/r(i)\end{aligned}$$

□

By theorem 5.4, expressions like $\boldsymbol{\varphi}(j)^T\mathbf{P}(k)\boldsymbol{\varphi}(i)$ are now computable using the kernel trick. Furthermore, we have the following theorem which converts the weight estimate into a linear combination of $\{\boldsymbol{\varphi}(\mathbf{c}_m)\}_{m=1}^M$.

Theorem 5.5. *By the assumption in equation* (5.37)*, the optimal state estimate in equation* (5.21) *can be expressed as*

$$\boldsymbol{\omega}(j) = \mathbf{C}\mathbf{a}(j) \tag{5.41}$$

where $\mathbf{a}(j)$ *is an* $M \times 1$ *real-valued vector, for all* $j > 0$.

Proof. Notice that by equation (5.21)

$$\begin{aligned}\mathbf{k}_p(1) &= \frac{\mathbf{C}\mathbf{B}\mathbf{h}_{\mathbf{E}}(1)}{\lambda r(1)} \\ \boldsymbol{\omega}(1) &= \mathbf{C}\left[\frac{\mathbf{B}\mathbf{h}_{\mathbf{E}}(1)d(1)}{\lambda r(1)}\right] = \mathbf{C}\mathbf{a}(1)\end{aligned}$$

Thus, the claim is valid for $j = 1$. Then, we use the mathematical induction to prove it is valid for any j. Assume it is true for $i - 1$. By the recursion in equation (5.21) and the result from Theorem 5.4, we have

$$\begin{aligned}\boldsymbol{\omega}(i) &= \mathbf{CBE}^T\boldsymbol{\omega}(i-1) + k_p(i)e(i)\\ &= \mathbf{CBG_{EC}a}(i-1) + \mathbf{CBE}^T\mathbf{P}(i-1)\boldsymbol{\varphi}(i)e(i)/r(i)\\ &= \mathbf{CB}\{\mathbf{G_{EC}a}(i-1) + \mathbf{E}^T[\rho(i-1)\mathbf{I} - \mathbf{CQ}(i-1)\mathbf{C}^T]\varphi(i)e(i)/r(i)\}\\ &= \mathbf{CB}\{\mathbf{G_{EC}a}(i-1) + [\rho(i-1)\mathbf{h_E}(i) - \mathbf{G_{EC}Q}(i-1)\mathbf{h_C}(i)]e(i)/r(i)\}\end{aligned}$$

Therefore

$$\mathbf{a}(i) = \mathbf{B}\{\mathbf{G_{EC}a}(i-1) + [\rho(i-1)\mathbf{h_E}(i) - \mathbf{G_{EC}Q}(i-1)\mathbf{h_C}(i)]e(i)/r(i)\}$$

This completes the proof. □

Hence, after representing $\boldsymbol{\omega}$ with $\mathbf{a}$, $\mathbf{P}$ with $\mathbf{Q}$, and noticing that $\rho(i) = \beta^i q$ is not needed in the recursion, we have Algorithm 13.

Algorithm 13. The extended kernel recursive least-squares algorithm with finite rank assumption [equation (5.37)]

Initialize

$$\mathbf{a}(1) = \frac{\mathbf{Bh_E}(1)d(1)}{\lambda\beta + \kappa(\mathbf{u}(1), \mathbf{u}(1))}$$

$$\mathbf{Q}(1) = -\frac{\mathbf{BG_EB}^T}{\lambda} + \frac{\mathbf{Bh_E}(1)\mathbf{h_E}(1)^T\mathbf{B}^T}{\lambda^2\beta + \lambda\kappa(\mathbf{u}(1), \mathbf{u}(1))}$$

Iterate for $i > 1$:

$$r(i) = \beta^i + \beta^{i-1}q\kappa(\mathbf{u}(i), \mathbf{u}(i)) - \mathbf{h_C}(i)^T\mathbf{Q}(i-1)\mathbf{h_C}(i)$$

$$e(i) = d(i) - \mathbf{h_C}(i)^T\mathbf{a}(i-1)$$

$$\mathbf{a}(i) = \mathbf{BG_{EC}a}(i-1) + \mathbf{B}[\beta^{i-1}q\mathbf{h_E}(i) - \mathbf{G_{EC}Q}(i-1)\mathbf{h_C}(i)]e(i)/r(i)$$

$$\begin{aligned}\mathbf{Q}(i) = &-\beta^{i-1}q\mathbf{BG_EB}^T - \mathbf{BG_{EC}Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T\\ &+ \mathbf{Bh_E}(i)\mathbf{h_E}(i)^T\mathbf{B}^T[\beta^{i-1}q]^2/r(i) + \mathbf{Bh_E}(i)\mathbf{h_C}(i)^T\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T\beta^{i-1}q/r(i)\\ &+ \mathbf{BG_{EC}Q}(i-1)\mathbf{h_C}(i)\mathbf{h_E}(i)^T\mathbf{B}^T\beta^{i-1}q/r(i)\\ &+ \mathbf{BG_{EC}Q}(i-1)\mathbf{h_C}(i)\mathbf{h_C}(i)^T\mathbf{Q}(i-1)\mathbf{G}_{\mathbf{EC}}^T\mathbf{B}^T/r(i)\end{aligned}$$

Note that throughout the iteration, $\mathbf{a}(i)$ is an $M \times 1$ vector and $\mathbf{Q}(i)$ is an $M \times M$ matrix. So again, the dimension of $\mathbf{a}(i)$ and $\mathbf{Q}(i)$ only depends on the dimensionality of $\mathrm{im}(\mathbf{A})$ regardless of the dimension of $\boldsymbol{\varphi}$. This algorithm can be regarded as a fixed-topology RBF network (comparing with the growing structure of Algorithm 11), and the linear coefficients $\mathbf{a}(i)$ is constantly updated based on the state transition process and the input-output observations. The time and space complexities are both $O(M^2)$ per iteration.

If the state noise $\mathbf{n}(i) \in \mathbb{F}$ is further assumed within the span of $\{\boldsymbol{\varphi}(\mathbf{c}_m)\}_{m=1}^M$, i.e.,

$$\mathbf{n}(i) = \mathbf{C}\tilde{\mathbf{n}}(i)$$

then it can be observed that the state $\mathbf{x}(i) \in \mathbb{F}$ also lies in this subspace for $i > 0$. Therefore,

$$\mathbf{x}(i) = \mathbf{C}\tilde{\mathbf{x}}(i)$$

Note that $\tilde{\mathbf{n}}(i)$ and $\tilde{\mathbf{x}}(i)$ are M-dimensional real-valued vectors.

With all these assumptions, the model in equation (5.16) reduces to

$$\begin{aligned} \tilde{\mathbf{x}}(i+1) &= \mathbf{B}\mathbf{G}_{\mathbf{EC}}\tilde{\mathbf{x}}(i) + \tilde{\mathbf{n}}(i) \\ d(i) &= \mathbf{h}_{\mathbf{C}}(i)^T \tilde{\mathbf{x}}(i) + v(i) \end{aligned} \tag{5.42}$$

and Algorithm 13 reduces to a classic (linear) Kalman recursion on state $\tilde{\mathbf{x}}(i)$, with $\{\mathbf{h}_{\mathbf{C}}(i), d(i)\}$ as the input–output observations. Although this result is not surprising with the subspace technique, it provides much understanding about state transition processes in the RKHS.

5.6 COMPUTER EXPERIMENTS

5.6.1 EX-KRLS Applied to Rayleigh Channel Tracking

Rayleigh fading is a statistical model for the effect of a propagation environment on a radio signal, such as that used by wireless devices. In wireless communications, multiple reflections of transmitted signals will arrive to a receiver because of complex communication channels. The reflections are with different amplitude and phase distortions, which may by purely random. The overall received signal is the combined sum of all the reflections. Based on the relative phases of the reflections, the signals may add up constructively or destructively at the receiver. Furthermore, the physical channel may be changing over time, for example, the receiver (e.g., a cell phone) is moving, so these destructive and constructive interferences will vary with time. This phenomenon is known as channel fading. Rayleigh fading is a reasonable model when many objects in the environment scatter the radio signal before it arrives at the receiver. By the central limit theorem, if there is sufficient scatter, the channel impulse response will be well modeled as a Gaussian process regardless of the distribution of the individual components. If there is no dominant component to the scatter, then such a process will have zero mean and phase evenly distributed between 0 and 2π radians. The envelope of the channel response will therefore be Rayleigh distributed. In other words, if the impulse response of a single-tap fading channel is

$$h(n) = x(n)\delta(n - \tau) \tag{5.43}$$

then the Rayleigh distribution states that

$$p(|x(n)|) = \frac{|x(n)|}{\sigma^2}\exp\left(\frac{-|x(n)|^2}{2\sigma^2}\right)$$
$$p(\angle x(n)) = \frac{1}{2\pi} \tag{5.44}$$

where $\{x(n)\}$ is a time-variant complex number that models the time-variations in the channel. $|x(n)|$ denotes magnitude and $\angle x(n)$ denotes phase. τ is the channel delay. σ is the mode of the distribution. The mean and variance can be expressed as

$$\mathbf{E}[|x(n)|] = \sigma\sqrt{\pi/2}$$
$$\mathbf{E}\left\{[|x(n)| - \mathbf{E}[|x(n)|]]^2\right\} = \frac{4-\pi}{2}\sigma^2 \tag{5.45}$$

Moreover, the normalized autocorrelation function with motion at a constant velocity is modeled by a zeroth-order Bessel function of the first kind

$$\mathbf{E}[x(n)x(n-k)] = \mathcal{J}_0(2\pi f_D T_s k), \quad k = \ldots, -1, 0, 1, \ldots \tag{5.46}$$

where T_s is the sampling period of the sequence, f_D is the maximum Doppler frequency of the Rayleigh fading channel, and the function $\mathcal{J}_0(\cdot)$ is defined by

$$\mathcal{J}_0(y) = \frac{1}{\pi}\int_0^{\pi}\cos(y\sin\theta)\,d\theta \tag{5.47}$$

The Doppler frequency f_D is related to the speed of the receiver relative to the transmitter v and the carrier frequency f_c as follows:

$$f_D = \frac{vf_c}{c} \tag{5.48}$$

where c is the speed of light, $c = 3 \times 10^8$ m/s.

We consider the problem of tracking a nonlinear Rayleigh fading multipath channel and compare the performance of the proposed EX-KRLS algorithm with the original KRLS. Also the performance of normalized LMS and EX-RLS is included for comparison.

The nonlinear Rayleigh fading multipath channel employed here is the cascade of a traditional Rayleigh fading multipath channel and a saturation nonlinearity. In the Rayleigh multipath fading channel, the number of the paths is chosen as $M = 5$, the maximum Doppler frequency $f_D = 100$ Hz, and the sampling rate $T_s = 0.8\,\mu$s (so it is a slow-fading channel with the same fading rate for all the paths). All the tap coefficients are generated according to the

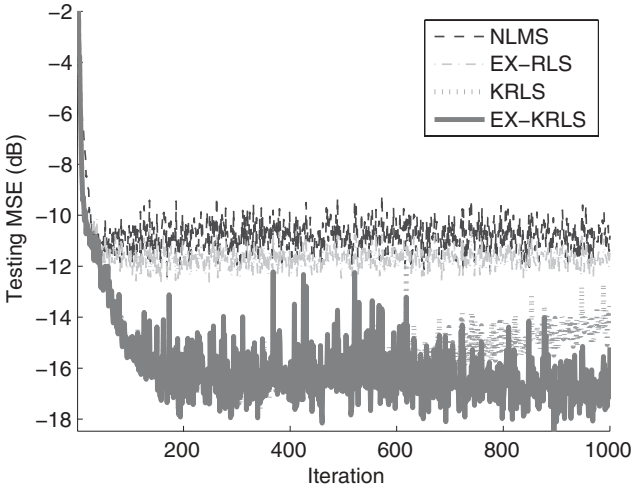


Figure 5.1. Ensemble learning curves of NLMS, EX-RLS, KRLS, and EX-KRLS in tracking a Rayleigh fading multipath channel.

Rayleigh model but only the real part is used in this experiment. A white Gaussian-distributed time series (with unit power) is sent through this channel, corrupted with the additive white Gaussian noise (with variance $\sigma^2 = 0.001$) and then the saturation nonlinearity $y = \tanh(x)$ is applied, where x is the output of the Rayleigh channel. The whole nonlinear channel is treated as a black box, and only the input and output are known.

Part 1: The tracking task is tested with four methods. The first one is normalized least-mean-squares algorithm (NLMS) (regularization factor $\varepsilon = 10^{-3}$, step-size parameter $\eta = 0.25$); the second one is EX-RLS ($\alpha = 0.999999937$, $q = 1.26 \times 10^{-7}$, $\beta = 0.995$, and $\lambda = 10^{-3}$).[3] The last two are nonlinear methods, namely KRLS (regularization parameter $\lambda = 0.01$) and the proposed EX-KRLS ($\mathbf{A} = \alpha\mathbf{I}$, $\alpha = 0.99999$, $q = 10^{-4}$, $\beta = 0.995$, and $\lambda = 0.01$).We use the Gaussian kernel in both cases with kernel parameter $a = 0.05$. Notice that α is close to 1 and q is close to 0 because the fading of the channel is slow. All the parameters are selected by scanning for best results.

We generate 1000 symbols for every experiment and perform 500 Monte Carlo experiments with independent inputs and additive noise. The ensemble learning curves are plotted in Figure 5.1, which show that EX-KRLS has a slightly better tracking ability than KRLS. The last 100 values in the learning curves are used to calculate the final mean square error (MSE), which is listed in Table 5.1. It is observed that the nonlinear methods outperform the linear methods significantly, because the channel model we use here is nonlinear. Although the Rayleigh channel is a slow-fading channel in this problem, we still enjoy a nearly 2.4-dB improvement by using EX-KRLS.

Part 2: Particle filters are sophisticated nonlinear model estimation techniques. The application of a particle filter requires the knowledge (or the

Table 5.1. Performance comparison of NLMS, EX-RLS, KRLS, and EX-KRLS in Rayleigh channel tracking.

Algorithm	MSE (dB)
NLMS	–10.85 ± 0.50
EX-RLS	–11.71 ± 0.32
KRLS	–14.41 ± 0.45
EX-KRLS	–16.82 ± 0.68

Table 5.2. Performance of particle filter in Rayleigh channel tracking.

Algorithm setting	MSE (dB)
Full knowledge	–31.09 ± 2.11
Random initial	–7.20 ± 5.00
$y = 0.9x + 0.1x^2$	2.56 ± 6.59
$y = tanh(x/2)$	–9.98 ± 4.76
$y = tanh(3x/2)$	–5.60 ± 5.28
$y = x$	2.50 ± 6.37

estimation) of the state and observation models. With only the input–output provided and assuming no knowledge about the process, there are no general approaches for using it in state estimation. To provide a reference (rather than a fair comparison), we test its performance with different assumptions.

The following model is assumed throughout the set of simulations

$$\begin{aligned} \mathbf{s}(i+1) &= \mathbf{F}\mathbf{s}(i) + \mathbf{n}(i) \\ d(i) &= h\left(\mathbf{s}(i)^T \mathbf{u}(i)\right) + \mathbf{v}(i) \end{aligned} \tag{5.49}$$

where $\mathbf{s}$ is regarded as the channel coefficients and h is the static nonlinearity.

In the first setting $\mathbf{F}$, the variance of $\mathbf{n}$ and $\mathbf{v}$ are estimated from the true channel value. Notice that this is not possible in practice. Nevertheless, we want to have the best possible performance from the particle filter. h is assumed known. The number of particles is set to 50. The true channel value at time 0 is used as the initial guess. The maximum a posterior estimation is used [Doucet et al., 2000b].

We also consider random initial conditions and model mismatch cases. For the random initial condition, we generate a standard normal distribution. For the model mismatch, we assume various models as listed in Table 5.2.

We compute the mean and standard deviation over 100 tests with independent inputs and initial states. The results are shown in Table 5.2. As we

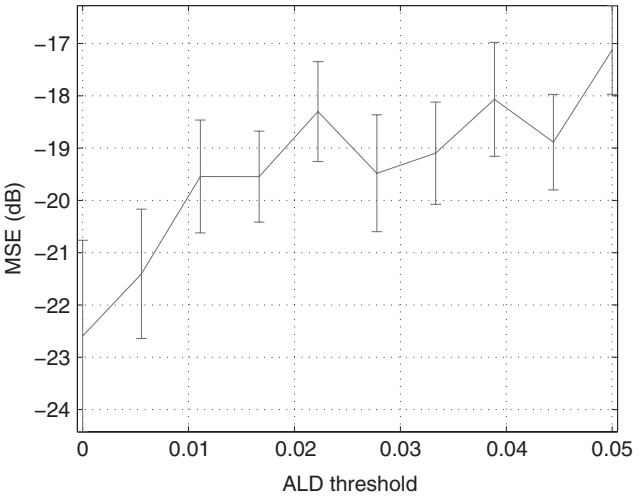


Figure 5.2. Performance of EX-KRLS by varying the threshold of ALD in Rayleigh channel tracking.

observe, the particle filter can outperform the extended KRLS with the full knowledge of the model. However, it is sensitive to the initialization, and it performs badly with an incorrect model.

Part 3: In the last simulation, the effectiveness of ALD to reduce the complexity in EX-KRLS is tested. We choose 10 thresholds in ALD in the range of [0, 0.05]. For each threshold, 500 symbols are generated for every experiment and 50 Monte Carlo experiments are conducted. The final MSE is calculated with the last 100 values in the ensemble learning curves. The final MSE versus the threshold is plotted in Figure 5.2. The final MSE versus the final network size is plotted in Figure 5.3. The regularization parameter is set as 0.01. The trade-off between complexity and performance by tuning the threshold in ALD is clearly depicted in Figure 5.3. The kernel parameter is set as 0.1 here.

5.6.2 EX-KRLS Applied to Lorenz Time-Series Prediction

The Lorenz attractor, introduced by Edward Lorenz in 1963 [Lorenz, 1963], is a dynamical system corresponding to the long-term behavior of a chaotic flow, and it is noted for its butterfly shape. The system is nonlinear, three dimensional, and deterministic. In 2001 it was proven by Warwick Tucker that for a certain set of parameters, the system exhibits chaotic behavior and displays what is today called a strange attractor [Tucker, 2002].

The following set of differential equations dictates how the state of Lorenz system evolves over time in a complex, nonrepeating pattern:

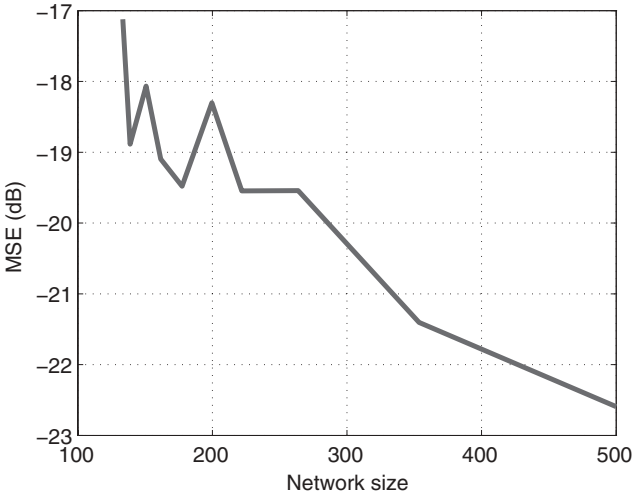


Figure 5.3. Network size versus MSE in EX-KRLS by varying the threshold of ALD in Rayleigh channel tracking.

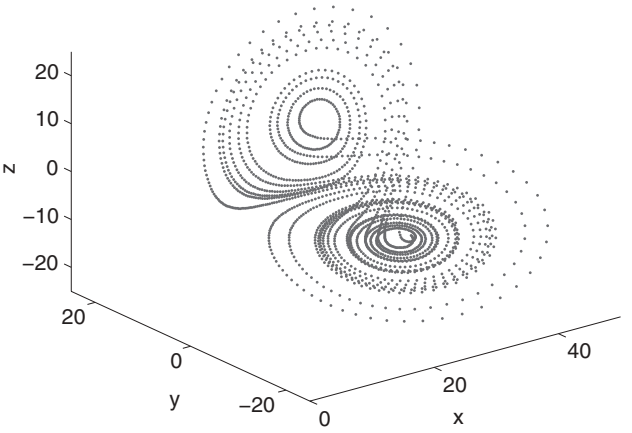


Figure 5.4. State trajectory of a Lorenz system with values $\beta = 8/3$, $\sigma = 10$, and $\rho = 28$.

$$\begin{aligned} \frac{dx}{dt} &= -\beta x + yz \\ \frac{dy}{dt} &= \sigma(z - y) \\ \frac{dx}{dt} &= -xy + \rho y - z \end{aligned} \tag{5.50}$$

Setting $\beta = 8/3$, $\sigma = 10$, and $\rho = 28$, the state evolution pattern is plotted in Figure 5.4. The first-order approximation is used with a step-size parameter 0.01.

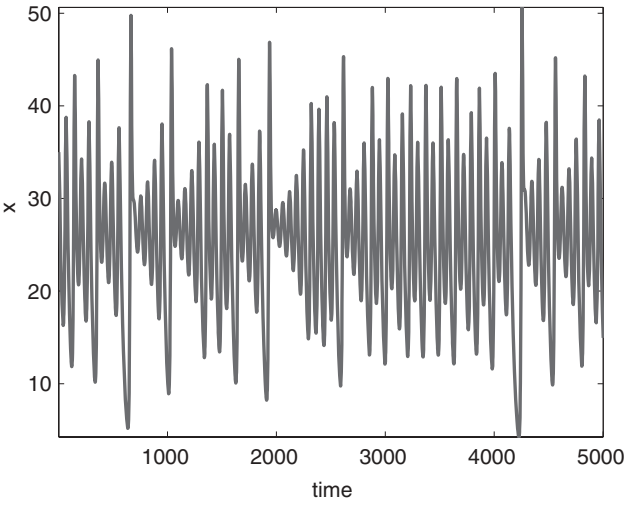


Figure 5.5. Typical waveform of the first component from the Lorenz system.

We pick the first component, namely x here, for the short-term prediction task. The first component is plotted in Figure 5.5.

The short-term prediction task can be formulated using five past data $\mathbf{u}(i) = [x(i-5), x(i-4), \ldots, x(i-1)]^T$ as the input to predict $x(i+T)$, which is the desired response here. T is the prediction step. The larger T is, the more nonlinearity the system exhibits and, hence, the harder the problem is. The signal is preprocessed to be zero mean and unit variance before modeling.

Part 1: In the first simulation, we test normalized LMS, RLS, EX-RLS, KRLS, and EX-KRLS on this task. We pick 20 prediction steps in the range of [1, 20]. For each prediction step, 50 Monte Carlo simulations are run with different segments of the signal. The length of each segment is 1000 points. The final MSE is calculated from the last 100 points in the ensemble learning curves, which are averaged over all the Monte Carlo simulations. The regularization factor $\varepsilon = 10^{-3}$, and the step-size parameter $\eta = 0.2$ in NLMS. The regularization parameter $\lambda = 10^{-3}$ in RLS. $\alpha = 1$, $q = 0.01$, $\beta = 0.99$, $\lambda = 10^{-3}$ in EX-RLS. $\lambda = 10^{-3}$ in KRLS and $\mathbf{A} = \alpha\mathbf{I}$, $\alpha = 1$, $q = 0.01$, $\beta = 0.99$, $\lambda = 10^{-3}$ in EX-KRLS. We use the Gaussian kernel with kernel parameter $a = 1$. The performance is plotted in Figure 5.6. It can be observed that the extended models exhibit better performance because the Lorenz system is switching between two attractors as shown in Figure 5.4. With small prediction steps, the linear models perform equally well to the nonlinear models because the sampling rate is high and the signal is smooth in Figure 5.5.

Part 2: It is noticed that there are a few parameters to tune in each algorithm. We experiment with different parameters and choose the best through scanning. In this simulation, we use cross-validation to select the best kernel parameter. Ten values of kernel parameter are picked in the interval of [0, 10]. With each kernel parameter, we train the EX-KRLS algorithm and record

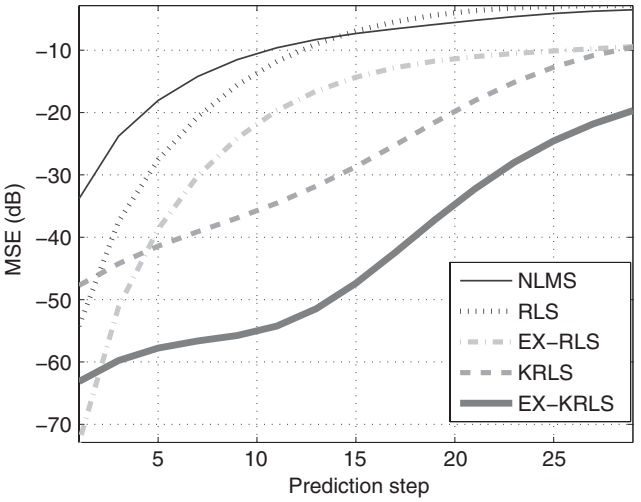


Figure 5.6. Performance comparison of NLMS, RLS, EX-RLS, KRLS, and EX-KRLS in Lorenz system prediction with different prediction steps.

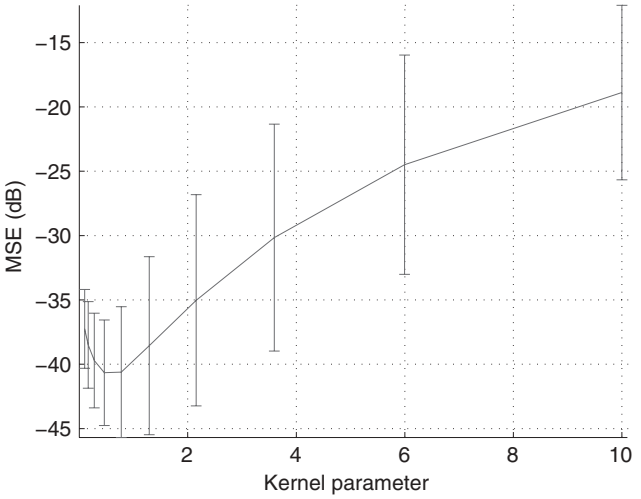


Figure 5.7. Effect of kernel parameter on performance of EX-KRLS in Lorenz system prediction.

the testing MSE. All the other settings are the same as in Part 1. The testing MSE versus the kernel parameter is plotted in Figure 5.7 (prediction step = 10). As we see, the best kernel parameter for this problem is around 0.8.

Part 3: In this simulation, we fix the prediction step at 10 and plot the ensemble learning curves of NLMS, EX-RLS, KRLS, and EX-KRLS. The settings of the algorithms are the same as in Part 1. We conduct 200 Monte Carlo

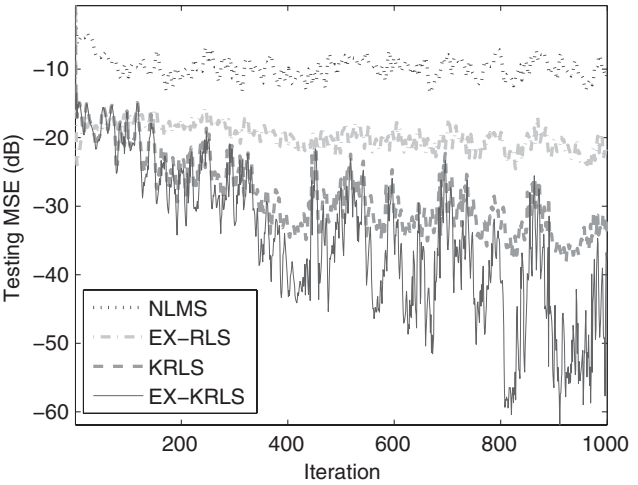


Figure 5.8. Ensemble learning curves of NLMS, EX-RLS, KRLS, and EX-KRLS in Lorenz system prediction (prediction step = 10).

Table 5.3. Performance comparison of NLMS, RLS, EX-RLS, KRLS, and EX-KRLS in Lorenz system prediction.

Algorithm	MSE (dB)
NLMS	−8.12 ± 0.91
RLS	−11.83 ± 0.71
EX-RLS	−19.89 ± 0.77
KRLS	−32.53 ± 1.13
EX-KRLS	−44.92 ± 4.80

simulations with different channel parameters and independent training data. In Figure 5.8, the ensemble learning curves are shown by averaging 200 Monte Carlo simulations. The final MSE is also listed in Table 5.3. As expected, the EX-KRLS has the best performance, followed by KRLS, EX-RLS, and NLMS.

Part 4: In the last simulation, we compare the performance of SW-KRLS with EX-KRLS. Because the modeling and tracking performance of SW-KRLS depends on the window length, we pick 20 different window lengths in the range of [10, 400]. For each window length, 50 Monte Carlo simulations are run with different segments of signal. Each segment has 500 points. The prediction step is fixed as 10. The performance is plotted in Figure 5.9. Clearly, EX-KRLS outperforms SW-KRLS significantly in this example. The trend of improving performance with longer window length shows SW-KRLS fails to improve the tracking ability because it eventually defaults to KRLS with the best performance.

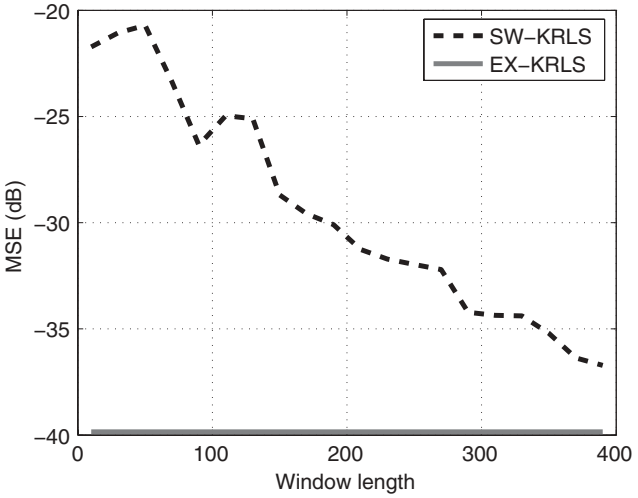


Figure 5.9. Performance comparison of SW-KRLS and EX-KRLS in Lorenz system prediction.

5.7 CONCLUSION

The extended kernel recursive least-squares algorithm is presented focusing on two special cases of the state transition matrix. Exponentially weighted KRLS and random-walk KRLS are also discussed as special cases, and all of them can be regarded as nontrivial extensions to KRLS, with an explicit state transition process. This algorithm is suitable to model nonlinear systems with a slow fading and a small variation in state. Compared with the existing KRLS algorithms, EX-KRLS is a step closer to implementing the kernel Kalman filters. Preliminary results of this algorithm are promising, which suggests it can have a wide applicability in nonlinear extensions of EX-RLS.

Examples of Rayleigh channel tracking and Lorenz system prediction are illustrated with comparison among EX-KRLS, KRLS, SW-KRLS, and linear methods. They show that EX-KRLS outperforms the others in terms of its modeling and tracking abilities. And more strikingly, the performance boost is achieved based on a simple tracking model. The particle filter can outperform EX-KRLS only with full knowledge of the model, which is usually impractical.

Apparently, how to learn the state transition model itself is important if such information is absent from domain knowledge. Although conventional methods such as maximum likelihood and expectation-maximization algorithm [Dempster et al., 1977, Ralaivola and d'Alche Buc, 2005] seem appropriate to the problem, a more detailed analysis should be conducted in the future. Also, how to derive a full-blown state estimation algorithm in RKHS still remains as one of the most fascinating problems to be conquered in this subject.

ENDNOTES

1. **Nonlinear Kalman Filters.** The Kalman filter is a powerful state estimation method for linear dynamic systems. However, most realistic system are nonlinear. The nonlinearity can be associated either with the state model, the observation model, or both.

   The extended Kalman filter [Sorenson, 1985] uses the first-order approximation (Jacobian matrix) to the nonlinearities in the step of covariance propagation. The unscented Kalman filter [Julier and Uhlmann, 1997] is an improvement to the extended Kalman filter by evaluating the posterior distribution with a set of sample points. The cubature Kalman filter [Arasaratnam and Haykin, 2009] uses the spherical-radial cubature rule to compute multivariate moment integrals numerically encountered in the nonlinear Bayesian filter. Other nonlinear Kalman filters employing local approximation include central-difference Kalman filter [Schei, 1997] and quadrature Kalman filter [Ito and Xiong, 2000].

   In addition, the particle filter uses Monte Carlo integrations with the importance sampling [Gordon et al., 1993]. Other nonlinear Kalman filters employing global approximation include point-mass filter [Simandl et al., 2006] and Gaussian mixture filter [Alspach and Sorenson, 1972].
2. For more details on the derivation of the extended RLS algorithm, please refer to Sayed [2003], pages 768–774.
3. For more details about the parameter setting in EX-RLS, please refer to Sayed [2003], page 759.

6

DESIGNING SPARSE KERNEL ADAPTIVE FILTERS

This chapter addresses the principal bottleneck of this class of online kernel algorithms, which is related to its growing structure with each new sample. The novelty criterion and approximate linear dependency test discussed in the previous chapters handle effectively the problem of growing structure, but they are heuristic in nature. Therefore, how to establish mathematically a framework to test whether a given sample is needed to improve performance is of great significance.

Toward this goal, an information theoretic measure called surprise is introduced here. Surprise captures the uncertainty of a new example with respect to the current knowledge of the learning system. Once the instantaneous information contained on a datum is defined, it is possible to estimate surprise directly from data using Gaussian process theory. This criterion allows us to discard or include new exemplars in the filter structure systematically and curb its growth. This information criterion provides a unifying view on and a solid mathematical foundation for novelty criterion and approximate linear dependency. And it also gives a general framework for redundancy removal, abnormality detection, and knowledge discovery.

6.1 DEFINITION OF SURPRISE

Surprise comes from observations but not any observation is surprising. For example, when knowledge of the weather is necessary, one watches a weather

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

forecast (a state transition from ignorance to being informed), but watching the same program for a second time does not improve our knowledge. The observations are exactly the same, but the effects are different. This is because the second observation comes with no surprise. Expansion of the boundary of knowledge is always preceded by surprise.

Obviously, this concept is important for learning and adaptation, but there is no widely accepted mathematical definition in the literature.[1] Suppose the learning machine is $y(\mathbf{u}; \mathcal{T}(i))$ after processing a set of training data $\mathcal{D}(i) = \{\mathbf{u}(j), d(j)\}_{j=1}^{i}$, where $\mathcal{T}(i)$ specifies the state of the learning system at time i. The problem is to measure how much information a new example $\{\mathbf{u}(i+1), d(i+1)\}$ contains that is "transferable to" the current learning system.

First, let us observe why the definition from the classic information theory does not apply here. According to Shannon [1948], the information contained on an exemplar $\{\mathbf{u}(i+1), d(i+1)\}$ is defined as

$$I(i+1) = -\ln p(\mathbf{u}(i+1), d(i+1)) \tag{6.1}$$

where $p(\mathbf{u}(i+1), d(i+1))$ is the true joint probability density. This definition is widely used and achieves huge successes in digital communications, game theory, and other fields [Cover and Thomas, 1991]. However, it has at least two problems in the learning setting. First of all, the learning machine never knows the true joint probability density, which is the ultimate knowledge meant to be learned by the machine [Poggio and Smale, 2003]. Second, it is observer independent. This is certainly undesirable in the context of learning because it cannot distinguish novelty from redundancy.

Because the true joint distribution is unknown and an observer-dependent information measure is sought, a natural idea is to define the information measure based on the posterior distribution hypothesized by the learning system.

Definition 1. *Surprise is a subjective information measure of an exemplar* $\{\mathbf{u}, d\}$ *with respect to a learning system* $\mathcal{T}$*. Denoted by* $S_{\mathcal{T}}(\mathbf{u}, d)$*, it is defined as the negative log likelihood of the exemplar given the learning system's hypothesis on the data distribution:*

$$S_{\mathcal{T}}(\mathbf{u}, d) = -\ln p(\mathbf{u}, d|\mathcal{T}) \tag{6.2}$$

where $p(\mathbf{u}, d|\mathcal{T})$ *is the subjective probability of* $\{\mathbf{u}, d\}$ *hypothesized by* $\mathcal{T}$*.*

$S_{\mathcal{T}}(\mathbf{u}, d)$ measures how "surprising" the exemplar is to the learning system. Applying this definition directly to the active online learning problem, we have the surprise of $\{\mathbf{u}(i+1), d(i+1)\}$ to the current learning system $\mathcal{T}(i)$ simply as

$$S_{\mathcal{T}(i)}(\mathbf{u}(i+1), d(i+1)) = -\ln p(\mathbf{u}(i+1), d(i+1)|\mathcal{T}(i)) \tag{6.3}$$

where $p(\mathbf{u}(i+1), d(i+1)|\mathcal{T}(i))$ is the posterior distribution of $\{\mathbf{u}(i+1), d(i+1)\}$ hypothesized by $\mathcal{T}(i)$. Denote $S(i+1) = S_{\mathcal{T}(i)}(\mathbf{u}(i+1), d(i+1))$ for simplicity in the following discussion.

Intuitively, if $p(\mathbf{u}(i+1), d(i+1)|\mathcal{T}(i))$ is large, the new datum $\{\mathbf{u}(i+1), d(i+1)\}$ is well expected by the learning system $\mathcal{T}(i)$ and thus contains a small amount of information to be learnt. However, if $p(\mathbf{u}(i+1), d(i+1)|\mathcal{T}(i))$ is small, then the new datum "surprises" the learning system, which means either the data contains something new for the system to discover or it is suspicious.

According to this measure, we can classify the new exemplar into the following three categories:

- *Abnormal*: $S(i+1) > T_1$.
- *Learnable*: $T_1 \geq S(i+1) \geq T_2$.
- *Redundant*: $S(i+1) < T_2$.

T_1, T_2 are problem-dependent parameters. The choice of the thresholds and learning strategies defines the characteristics of the learning system.

6.2 A REVIEW OF GAUSSIAN PROCESS REGRESSION

It is a difficult problem in general to estimate the posterior distribution. One way is to use kernel density estimation as in Dima and Hebert [2005], but this is problematic when the dimensionality of $\mathbf{u}$ is high. Another way is to convert it to a parameter estimation problem by assuming a parametric distribution family. Itti and Baldi [2006] studied an image understanding task by assuming a Poisson distribution across the models. Here we use Gaussian processes theory.

Denote the output of the learning system as $y(i)$ given the input $\mathbf{u}(i)$, for any i. By the Gaussian process (GP) theory, the prior distribution of outputs of the learning system given all the inputs are assumed to be jointly Gaussian, i.e.,

$$p(y(1), \cdots, y(i)|\mathbf{u}(1), \cdots \mathbf{u}(i)) = \mathcal{N}\left(0, \sigma_n^2\mathbf{I} + \mathbf{G}(i)\right) \tag{6.4}$$

where

$$\mathbf{G}(i) = \begin{bmatrix} \kappa(\mathbf{u}(1), \mathbf{u}(1)) & \cdots & \kappa(\mathbf{u}(i), \mathbf{u}(1)) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{u}(1), \mathbf{u}(i)) & \cdots & \kappa(\mathbf{u}(i), \mathbf{u}(i)) \end{bmatrix}$$

for any i. σ_n^2 is the variance of the noise contained in the observation. κ is the covariance function. It is symmetric and positive definite. As a matter of fact,

a covariance function is equivalent to a reproducing kernel extensively used in this book. As we have observed, the commonly used covariance is the Gaussian kernel

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp\left(-a\|\mathbf{u} - \mathbf{u}'\|^2\right) \tag{6.5}$$

The idea behind this seemingly naive assumption is as follows:

- For any input $\mathbf{u}$, the prior distribution of the output $y(\mathbf{u})$ is Gaussian with zero mean and variance $\kappa(\mathbf{u}, \mathbf{u}) + \sigma_n^2$, where the component σ_n^2 comes from the noise.
- For any two inputs $\mathbf{u}$, $\mathbf{u}'$, the a priori distribution of the outputs $y(\mathbf{u})$ and $y(\mathbf{u}')$ are jointly Gaussian. The correlation between the two is determined by $\kappa(\mathbf{u}, \mathbf{u}')$. It is clear that if the Gaussian kernel [equation (6.5)] is used, the closer the two inputs are, the stronger correlation the two outputs have, i.e., an explicit smoothness constraint is imposed. Notice that this assumption provides the basis for inference just as the similarity measure defined by the kernel function in RKHS.
- For multiple inputs, the same idea applies.

As we know from Bayesian inference theory [Duda et al., 2000], the a priori distribution is only important when there is insufficient data. In practice, a naïve prior such as Gaussian or uniform gives good inference results as the posterior converges quickly. Other non-Gaussian priors are also possible, but the price to pay is a much higher computational complexity.

The posterior distribution of the output $y(i + 1)$ given the input $\mathbf{u}(i + 1)$, and all past observations $\mathcal{D}(i)$ can be derived by the Bayes rule:

$$\begin{aligned} & p(y(i+1)|\mathbf{u}(i+1), \mathcal{D}(i)) \\ &= p(y(i+1)|y(1), \cdots, y(i), \mathbf{u}(1), \cdots, \mathbf{u}(i), \mathbf{u}(i+1)) \\ &= \frac{p(y(1), \cdots, y(i), y(i+1)|\mathbf{u}(1), \cdots, \mathbf{u}(i), \mathbf{u}(i+1))}{p(y(1), \cdots, y(i)|\mathbf{u}(1), \cdots, \mathbf{u}(i), \mathbf{u}(i+1))} \\ &= \frac{p(y(1), \cdots, y(i), y(i+1)|\mathbf{u}(1), \cdots, \mathbf{u}(i), \mathbf{u}(i+1))}{p(y(1), \cdots, y(i)|\mathbf{u}(1), \cdots, \mathbf{u}(i))} \end{aligned} \tag{6.6}$$

By the prior assumption in equation (6.4) and straightforward calculus, we have

$$p(y(i+1)|\mathbf{u}(i+1), \mathcal{D}(i)) = \mathcal{N}\left(\bar{d}(i+1), \sigma^2(i+1)\right) \tag{6.7}$$

which is again normally distributed, with

$$\bar{d}(i+1) = \mathbf{h}(i+1)^T \left[\sigma_n^2 \mathbf{I} + \mathbf{G}(i)\right]^{-1} \mathbf{d}(i) \tag{6.8}$$

$$\sigma^2(i+1) = \sigma_n^2 + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}(i+1)^T \left[\sigma_n^2 \mathbf{I} + \mathbf{G}(i)\right]^{-1} \mathbf{h}(i+1) \tag{6.9}$$

where

$$\mathbf{h}(i+1) = [\kappa(\mathbf{u}(i+1), \mathbf{u}(1)), \cdots, \kappa(\mathbf{u}(i+1), \mathbf{u}(i))]^T$$
$$\mathbf{d}(i) = [d(1), \cdots, d(i)]^T$$

The output of the learning system $y(i + 1)$ is a random variable by definition. $d(i + 1)$ is the actual observation (realization) of the output. So $p(y(i + 1)|\mathbf{u}(i + 1), \mathcal{D}(i))$ is a distribution (a function) and $p(d(i + 1)|\mathbf{u}(i + 1), \mathcal{D}(i))$ is the probability density at point $d(i + 1)$ (a value).

6.3 COMPUTING SURPRISE

Let us assume $\mathcal{T}(i) = \mathcal{D}(i)$ for now, i.e. the learning system memorizes all the past input–output pairs. Therefore, the posterior probability density $p(\mathbf{u}(i + 1), d(i + 1)|\mathcal{T}(i))$ can be evaluated by

$$\begin{aligned} p(\mathbf{u}(i+1), d(i+1)|\mathcal{T}(i)) &= p(d(i+1)|\mathbf{u}(i+1), \mathcal{T}(i))\, p(\mathbf{u}(i+1)|\mathcal{T}(i)) \\ &= \frac{1}{\sqrt{2\pi}\sigma(i+1)} \exp\left(-\frac{\left(d(i+1) - \bar{d}(i+1)\right)^2}{2\sigma^2(i+1)} \right) p(\mathbf{u}(i+1)|\mathcal{T}(i)) \end{aligned}$$

and therefore the surprise measure is

$$\begin{aligned} S(i+1) &= -\ln[p(\mathbf{u}(i+1), d(i+1)|\mathcal{T}(i))] \\ &= \ln\sqrt{2\pi} + \ln\sigma(i+1) + \frac{\left(d(i+1) - \bar{d}(i+1)\right)^2}{2\sigma^2(i+1)} - \ln[p(\mathbf{u}(i+1)|\mathcal{T}(i))] \end{aligned} \quad (6.10)$$

Equation (6.10) gives a whole picture of what factors and how these factors affect the surprise measure of the new datum. Some observations are as follows:

1. Surprise is proportional to the magnitude of the prediction error. $e(i + 1) = d(i + 1) - \bar{d}(i + 1)$ is the prediction error since $\bar{d}(i + 1)$ is the maximum a posterior (MAP) estimation of $d(i + 1)$ by the current learning system $\mathcal{T}(i)$. If $e^2(i + 1)$ is small, which means the learning system predicts well near $\mathbf{u}(i + 1)$, the corresponding $S(i + 1)$ is small.
2. Surprise is proportional to the prediction variance if the prediction error is small. In the case of a small prediction error, say $e(i + 1) \approx 0$, the second term $(d(i + 1) - \bar{d}(i + 1))^2/2\sigma^2(i + 1)$ is ineffective. And the $S(i + 1)$ is directly proportional to $\sigma(i + 1)$. A large variance indicates the learning system is uncertain about its guess even when the guess happens to be right. Incorporating the new datum will boost the prediction confidence near the neighborhood of the new datum in the future inference.

3. Surprise is huge with a small variance and a large prediction error. With a large prediction error and a small variance, the second term $(d(i+1) - \bar{d}(i+1))^2/2\sigma^2(i+1)$ dominates the measurement. This is a strong indication of abnormality: the machine is sure about its prediction but its prediction is far away from the observation. It exactly means the current observation is contradictory to what the machine had learnt before. Mathematically it may lead to instability if the machine attempts to incorporate the abnormal observation into the network.
4. A rare occurrence means more surprise. A smaller $p(\mathbf{u}(i+1)|\mathcal{T}(i))$ leads to a larger $S(i+1)$.

6.3.1 Input Distribution

The distribution $p(\mathbf{u}(i+1)|\mathcal{T}(i))$ is problem dependent. In the regression model, it is reasonable to assume

$$p(\mathbf{u}(i+1)|\mathcal{T}(i)) = p(\mathbf{u}(i+1)) \tag{6.11}$$

that is, the distribution of $\mathbf{u}(i+1)$ is independent of the previous observations or memoryless.

If the input has a normal distribution $\mathcal{N}(\mu, \Sigma)$, we have

$$S(i+1) = \ln\sigma(i+1) + \frac{\left(d(i+1) - \bar{d}(i+1)\right)^2}{2\sigma^2(i+1)} + \frac{(\mathbf{u}(i+1)-\mu)^T \Sigma^{-1} (\mathbf{u}(i+1)-\mu)}{2} \tag{6.12}$$

ignoring other constant terms.

In general, we can assume the distribution $p(\mathbf{u}(i+1))$ is uniform if no a priori information is available. Therefore, by discarding the constant terms, the surprise measure is simplified to

$$S(i+1) = \ln\sigma(i+1) + \frac{\left(d(i+1) - \bar{d}(i+1)\right)^2}{2\sigma^2(i+1)} \tag{6.13}$$

This formula is what we usually use in practice. $\bar{d}(i+1)$ and $\sigma(i+1)$ are defined in equation (6.8) and equation (6.9), respectively.

6.3.2 UNKNOWN DESIRED SIGNAL

The surprise measure depends on the knowledge of the desired signal $d(i+1)$. In the case of unknown desired signal, we simply average $S(i+1)$ over the posterior distribution of $y(\mathbf{u}(i+1))$.

By using equation (6.10), we have

$$\begin{aligned}\bar{S}(i+1) &= \int S(i+1)\, p(y|\mathbf{u}(i+1), \mathcal{T}(i))\, dy \\ &= \ln\sqrt{2\pi} + \ln\sigma(i+1) + \frac{1}{2} - \ln[p(\mathbf{u}(i+1)|\mathcal{T}(i))]\end{aligned}$$

Neglecting the constant terms yields

$$\bar{S}(i+1) = \ln\sigma(i+1) - \ln[p(\mathbf{u}(i+1)|\mathcal{T}(i))] \tag{6.14}$$

Furthermore, under memoryless uniform input assumption, it simplifies to

$$\bar{S}(i+1) = \ln\sigma(i+1) \tag{6.15}$$

In other words, the surprise measure defaults to the approximate linear dependency (ALD) or the variance measure in this case. This result explains rigorously why distance- or variance-based criteria are valid. Obviously, it does not use the important information contained in the desired signal.

6.3.3 The Probability of Novelty

Assume that the probability density function of the prediction error is $p(e)$, which does not contain any delta function at 0. According to the surprise criterion, the probability of accepting $\mathbf{u}(i+1)$ into the dictionary is

$$\begin{aligned}&P(\mathbf{u}(i+1) \text{ is accepted into the dictionary}) \\ &= P\left(T_1 \geq -\ln\left(\frac{1}{\sqrt{2\pi}\sigma(i+1)}\exp\left(-\frac{e(i+1)^2}{2\sigma^2(i+1)}\right)\right) \geq T_2\right) \\ &= P\left(2\sigma^2(i+1)\left(T_1 - \ln\sigma(i+1) - \ln\sqrt{2\pi}\right) \geq e^2(i+1) \geq 2\sigma^2(i+1)\right. \\ &\quad \left.\left(T_2 - \ln\sigma(i+1) - \ln\sqrt{2\pi}\right)\right) \\ &\leq \int_{-\sqrt{2\sigma^2(i+1)\left(T_1 - \ln\sigma(i+1) - \ln\sqrt{2\pi}\right)}}^{\sqrt{2\sigma^2(i+1)\left(T_1 - \ln\sigma(i+1) - \ln\sqrt{2\pi}\right)}} p(e)\, de\end{aligned} \tag{6.16}$$

when $T_1 - \ln\sigma(i+1) - \ln\sqrt{2\pi} > 0$, or equivalently $\sigma < \dfrac{\exp(T_1)}{\sqrt{2\pi}}$. Otherwise, the probability is 0.

The key point here is

$$\lim_{\sigma(i+1)\to 0} 2\sigma^2(i+1)\left(T_1 - \ln\sigma(i+1) - \ln\sqrt{2\pi}\right) = 0$$

In other words,

$$\lim_{\sigma(i+1)\to 0} P(\mathbf{u}(i+1) \text{ is accepted into the dictionary}) = 0$$

Therefore, the probability of being accepted into the dictionary is small if $\sigma(i+1)$ is small. In this sense, we say the prediction standard deviation $\sigma(i+1)$

is probabilistically lower bounded in online learning algorithms with the surprise criterion.

In the next section, we focus on how to design surprise-based sparse kernel adaptive filters.

6.4 KERNEL RECURSIVE LEAST SQUARES WITH SURPRISE CRITERION (KRLS-SC)

It is easy to verify that $\overline{d}(i+1)$ in equation (6.8) and $\sigma^2(i+1)$ in equation (6.9) equal $f_i(\mathbf{u}(\mathrm{i}+1))$ and $r(i+1)$, respectively, in KRLS with $\sigma_n^2 = \lambda$. Therefore, the surprise criterion can be integrated into KRLS seamlessly, and we call the algorithm KRLS-SC. The system starts with $f_1 = \mathbf{a}(1)\kappa(\mathbf{u}(1),\cdot)$ with $\mathbf{a}(1) = \mathbf{Q}(1)d(1)$, $\mathbf{Q}(1) = [\lambda + \kappa(\mathbf{u}(1), \mathbf{u}(1))]^{-1}$ and $\mathcal{C}(1) = \{\mathbf{c}_1 = \mathbf{u}(1)\}$. Then, it iterates the following procedure for $i \geq 1$:

For a new datum $\{\mathbf{u}(i+1), d(i+1)\}$, the following quantities are computed:

$$\begin{aligned}
\mathbf{h}(i+1) &= [\kappa(\mathbf{u}(i+1), \mathbf{c}_1), \cdots, \kappa(\mathbf{u}(i+1), \mathbf{c}_{m_i})]^T \\
f_i(\mathbf{u}(i+1)) &= \mathbf{h}(i+1)^T \mathbf{a}(i) \\
e(i+1) &= d(i+1) - f_i(\mathbf{u}(i+1)) \\
r(i+1) &= \lambda + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}(i+1)^T \mathbf{Q}(i)\mathbf{h}(i+1)
\end{aligned}$$

and the surprise measure

$$S(i+1) = \frac{1}{2}\ln r(i+1) + \frac{e^2(i+1)}{2r(i+1)} - \ln[p(\mathbf{u}(i+1)|\mathcal{T}(i))]$$

where $p(\mathbf{u}(i+1)|\mathcal{T}(i))$ can be assumed to be constant if no a priori information is available. As we observe, the computation of surprise comes with no additional complexity.

Based on this surprise measure, we can decide whether the example is abnormal, learnable, or redundant using thresholds. If the sample is abnormal or redundant, then it can be simply thrown away. If it is learnable, the system is updated by the standard KRLS algorithm, i.e.,

$$\mathcal{C}(i+1) = \{\mathcal{C}(i), \mathbf{u}(i+1)\} \tag{6.17}$$

$$\mathbf{z}(i+1) = \mathbf{Q}(i)\mathbf{h}(i+1) \tag{6.18}$$

$$\mathbf{Q}(i+1) = \begin{bmatrix} \mathbf{Q}(i) + \mathbf{z}(i+1)\mathbf{z}(i+1)^T / r(i+1) & -\mathbf{z}(i+1)/r(i+1) \\ -\mathbf{z}(i+1)^T / r(i+1) & 1/r(i+1) \end{bmatrix} \tag{6.19}$$

$$\mathbf{a}(i+1)=\begin{bmatrix}\mathbf{a}(i)-\mathbf{z}(i+1)e(i+1)/r(i+1)\\ e(i+1)/r(i+1)\end{bmatrix} \tag{6.20}$$

The updating rule is consistent with the observations in equation (6.10). On the one hand, if the prediction error is small, then the modifying quantities are small. In the extreme case, a redundant datum leads to negligible update. On the other hand, a large prediction error and a small prediction variance result in a large modification to the coefficients. In this extreme case, an abnormal datum causes instability. The overall complexity of KRLS-SC is $O(m_i^2)$ at iteration i.

6.5 KERNEL LEAST MEAN SQUARE WITH SURPRISE CRITERION (KLMS-SC)

The complexity of computing the exact surprise for KLMS is $O(m_i^2)$ at iteration i, which may offset the advantage of KLMS over other kernel adaptive filters in terms of simplicity. As we know, the surprise involves two basic concepts: the prediction error and the prediction variance. It is relatively easy to get the prediction error and the question is how we simplify the computation of the prediction variance. The approximation is based on the relationship between the prediction variance $\sigma^2(i+1)$ and the distance measure dis_2 in ALD. To simplify the computation of dis_2, we use the following distance measure as an approximation

$$\text{dis}_3 = \min_{\forall b, \forall \mathbf{c}_j \in \mathcal{C}(i)} \left\| \varphi(\mathbf{u}(i+1)) - b\varphi(\mathbf{c}_j) \right\| \tag{6.21}$$

i.e., selecting the "nearest" center in the dictionary to estimate the overall distance just like in the novelty criterion. By straightforward calculus, we have

$$\text{dis}_3^2 = \min_{\forall \mathbf{c}_j \in \mathcal{C}(i)} \left[\kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \frac{\kappa^2(\mathbf{u}(i+1), \mathbf{c}_j)}{\kappa(\mathbf{c}_j, \mathbf{c}_j)} \right]$$

When κ is a radial-basis function, this distance measure is equivalent to dis_1 used in the novelty criterion. And we can add a regularization term simply by including λ. Using the same notation from KRLS-SC, we have

$$r(i+1) = \lambda + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \max_{\forall \mathbf{c}_j \in \mathcal{C}(i)} \frac{\kappa^2(\mathbf{u}(i+1), \mathbf{c}_j)}{\kappa(\mathbf{c}_j, \mathbf{c}_j)} \tag{6.22}$$

Its complexity is $O(m_i)$ which is acceptable to KLMS. Therefore, we have the following KLMS-SC. The system starts with $f_1 = \mathbf{a}(1)\kappa(\mathbf{u}(1), \cdot)$ with $\mathbf{a}(1) = \eta d(1)$ and $\mathcal{C}(1) = \{\mathbf{c}_1 = \mathbf{u}(1)\}$. Then, it iterates the following procedure for $i \geq 1$:

For a new datum $\{\mathbf{u}(i+1), d(i+1)\}$, the following quantities are computed:

$$\begin{aligned}
\mathbf{h}(i+1) &= [\kappa(\mathbf{u}(i+1), \mathbf{c}_1), \cdots, \kappa(\mathbf{u}(i+1), \mathbf{c}_{m_i})]^T \\
f_i(\mathbf{u}(i+1)) &= \mathbf{h}(i+1)^T \mathbf{a}(i) \\
e(i+1) &= d(i+1) - f_i(\mathbf{u}(i+1)) \\
r(i+1) &= \lambda + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \max_{\forall \mathbf{c}_j \in \mathcal{C}(i)} \frac{\kappa^2(\mathbf{u}(i+1), \mathbf{c}_j)}{\kappa(\mathbf{c}_j, \mathbf{c}_j)}
\end{aligned}$$

The surprise measure is

$$S(i+1) = \frac{1}{2}\ln r(i+1) + \frac{e^2(i+1)}{2r(i+1)} - \ln[p(\mathbf{u}(i+1)|\mathcal{T}(i))]$$

where $p(\mathbf{u}(i+1)|\mathcal{T}(i))$ can be assumed to be constant if no a priori information is available. If the example is learnable, then the system is updated by the standard KLMS algorithm.

$$\mathcal{C}(i+1) = \{\mathcal{C}(i), \mathbf{u}(i+1)\} \tag{6.23}$$

$$\mathbf{a}_{i+1}(i+1) = \eta e(i+1) \tag{6.24}$$

The overall complexity of KLMS-SC is $O(m_i)$ at iteration i.

6.6 KERNEL AFFINE PROJECTION ALGORITHMS WITH SURPRISE CRITERION (KAPA-SC)

As we see, KLMS-SC uses only the "nearest" sample in the dictionary to approximate the distance. A natural idea is to use multiple (say K) centers to improve the approximation. Putting it into an optimization equation, we have

$$\text{dis}_4 = \min_{\forall \mathbf{b}, \forall \mathbf{n}_j \in \mathcal{C}(i)} \left\| \varphi(\mathbf{u}(i+1)) - \sum_{j=1}^{K} b_j \varphi(\mathbf{n}_j) \right\| \tag{6.25}$$

i.e., selecting K centers in the dictionary to estimate the overall distance. This is a combinatorial optimization scaling with $O(C_{m_i}^K K^3)$, where $C_{m_i}^K$ denotes the number of K combinations from the dictionary. A more feasible alternative is to select the K "nearest" neighbors one by one simply based on equation (6.21), whose complexity is $O(Km_i)$. After selecting the neighbors, the surprise measure can be computed and the learning system can be updated by the KAPA algorithms based on the K selected neighbors. We use KAPA-1 here as an example to illustrate the main steps involved to integrate the surprise criterion into KAPA

$$\mathbf{h}(i+1) = [\kappa(\mathbf{u}(i+1), \mathbf{c}_1), \cdots, \kappa(\mathbf{u}(i+1), \mathbf{c}_{m_i})]^T$$
$$f_i(\mathbf{u}(i+1)) = \mathbf{h}(i+1)^T \mathbf{a}(i)$$
$$e(i+1) = d(i+1) - f_i(\mathbf{u}(i+1))$$
$$r(i+1) = \lambda + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) - \mathbf{h}_{\mathbf{u}}^T [\mathbf{G}_{\mathbf{n}} + \lambda \mathbf{I}]^{-1} \mathbf{h}_{\mathbf{u}}$$
$$S(i+1) = \frac{1}{2} \ln r(i+1) + \frac{e^2(i+1)}{2r(i+1)} - \ln[p(\mathbf{u}(i+1)|\mathcal{T}(i))]$$

where

$$\mathbf{h}_{\mathbf{u}} = [\kappa(\mathbf{u}(i+1), \mathbf{n}_1), \cdots, \kappa(\mathbf{u}(i+1)), \mathbf{n}_K]^T$$
$$\mathbf{G}_{\mathbf{n}} = \begin{bmatrix} \kappa(\mathbf{n}_1, \mathbf{n}_1) & \cdots & \kappa(\mathbf{n}_1, \mathbf{n}_K) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{n}_K, \mathbf{n}_1) & \cdots & \kappa(\mathbf{n}_K, \mathbf{n}_K) \end{bmatrix}$$

If the datum is determined to be learnable, the system is updated in the following way:

$$e(i+1; k) = d(\mathbf{n}_k) - f_i(\mathbf{n}_k) \quad [\text{for} \quad 1 \le k \le K]$$
$$f_{i+1} = f_i + \eta e(i+1)\kappa(\mathbf{u}(i+1), \cdot) + \eta \sum_{k=1}^{K} e(i+1; k)\kappa(\mathbf{n}_k, \cdot)$$

where $d(\mathbf{n}_k)$ is the target associated with $\mathbf{n}_k$. $\{\mathbf{n}_k\}_{k=1}^K$ are the selected neighbors from the dictionary. We call the above algorithm KAPA-1-SC, which is different from the original KAPA-1 discussed in Chapter 3 and demands more computational resources ($O(Km_i + K^3)$). KAPA-1 uses the most K recent data points to approximate the gradient vector, whereas KAPA-1-SC uses the K "nearest" neighbors. It is argued that using the neighbors is more effective in terms of modeling, whereas using the most recent ones is better at tracking. In principle, approximating the prediction variance and approximating the gradient direction are independent and can be dealt with differently. For example, we can simply use the approximation in KLMS-SC to compute the surprise in KAPA without modification. We shall emphasize here that the real advantage of KAPA is its flexibility in design, and users should tailor the algorithm accordingly in practice.

6.7 COMPUTER EXPERIMENTS

6.7.1 Surprise Criterion Applied to Nonlinear Regression

In the example, we use a simple nonlinear regression problem to illustrate the main idea of surprise and its effectiveness in learning. The input-output

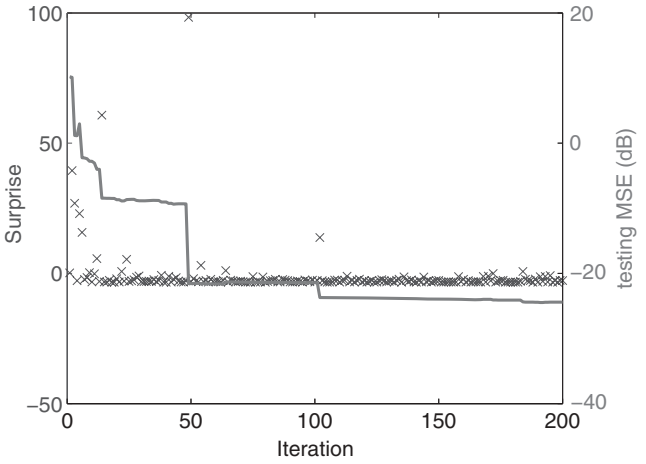


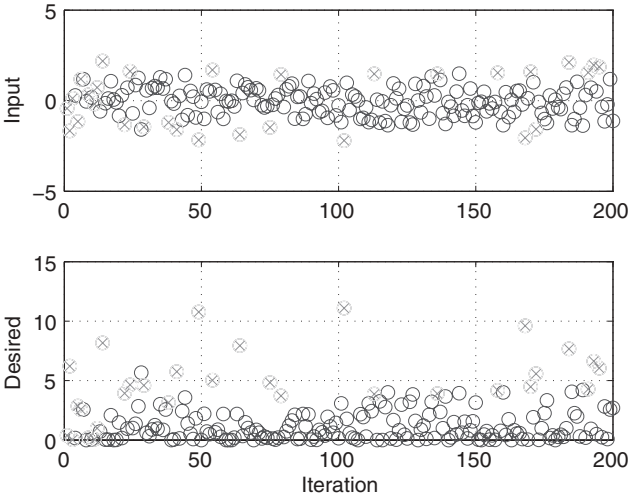
Figure 6.1. Surprise measure of training data (cross) and corresponding learning curve (solid linear) in nonlinear regression.

mapping is $y = -x + 2x^2 + \sin x$, and the input is Gaussian distributed with zero mean and unit variance.

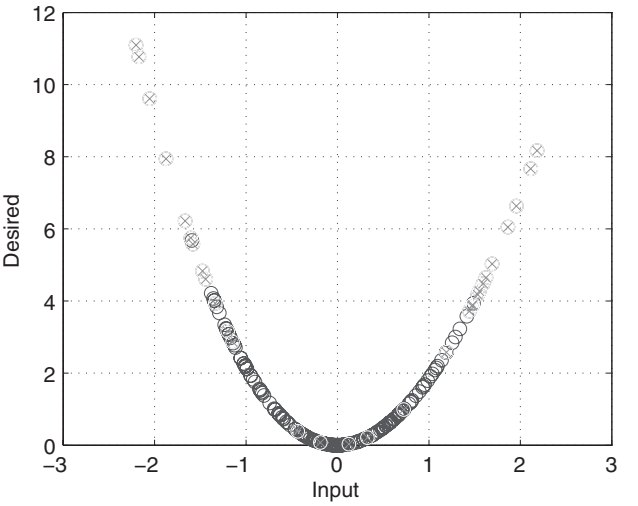
Part 1: In the first simulation, we use KRLS-SC and assume all data are learnable. The surprise is computed for every point during training. In general, the surprise measure of each datum depends on the relative order by which it is presented to the learning system. There are 200 points for training and 100 for testing. Figure 6.1 is a typical learning curve with mean-square error (MSE) calculated at each iteration on the testing data. It clearly shows that decreases in testing MSE (solid) directly result from learning informative data (crosses). The Gaussian kernel is used with $a = 0.2$. The regularization parameter $\lambda = 0.001$. The parameters are selected through cross-validation.

We also plot the 5% most "informative" data in Figure 6.2, which exhibits the characteristic of active sampling with few selected data in the well-defined region and more selected data near the boundary. The surprise measure is effective to distinguish novelty from redundancy.

Part 2: In the second simulation, we show how effective the method is to remove redundancy. We compare the surprise criterion (SC) [equation (6.12)] with the ALD test in KRLS. Notice that we use equation (6.15) here such that the two criteria have roughly the same scale. We also use T_2 to denote the threshold used in ALD. We test both KRLS-SC and KRLS-ALD with 50 different thresholds of T_2. A large T_1 is used to disable the abnormality detection. For each T_2, 100 Monte Carlo simulations are conducted with independent inputs to calculate the average number of centers and the corresponding average testing MSE. For each Monte Carlo simulation, 200 training points and 100 testing points are used. The result is illustrated in Figures 6.3, 6.4, and



A. The 5% most informative training data (cross with gray circle) along training iteration.



B. The 5% most informative training data (cross with gray circle) in two-dimensional space.

Figure 6.2. Informative training data according to the surprise measure in nonlinear regression.

6.5. It is clear that SC is effective with T_2 in a wide range of [–3, 5]. Even though ALD is equally effective with T_2 in the range of [–3, –2], a larger T_2 leads to catastrophic results (almost every points are excluded except the first one). By contrast, SC provides a balance by checking the prediction error. As

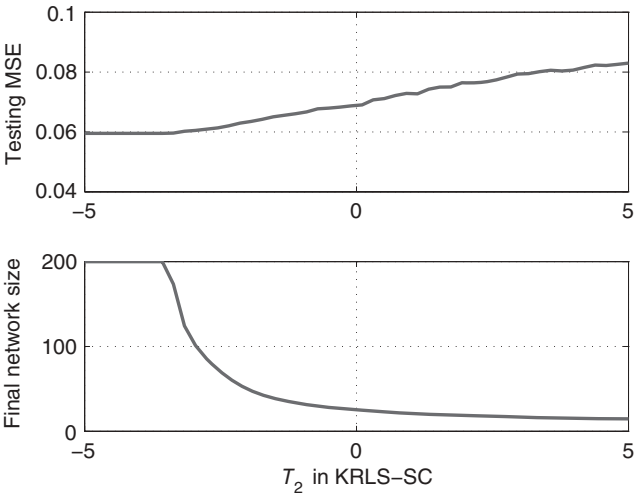


Figure 6.3. Final network size versus testing MSE of KRLS-SC by varying T_2 in nonlinear regression.

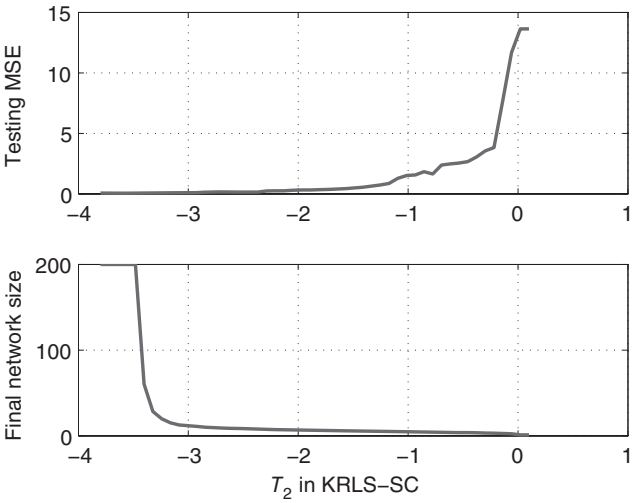


Figure 6.4. Final network size versus testing MSE of KRLS-ALD by varying T_2 in nonlinear regression.

shown in Figure 6.5, KRLS-SC is superior to KRLS-ALD. For the same MSE, KRLS-SC requires 10 to 20 less centers on average.

Part 3: In the third simulation, we show how KRLS-SC can be used to detect outliers whereas KRLS-ALD cannot. Two hundred training data are generated as before but 15 outliers are added manually at time indices 50, 60, …, 190 (by flipping their signs). We choose $T_2 = -3.14$, $T_1 = 200$ in SC, and

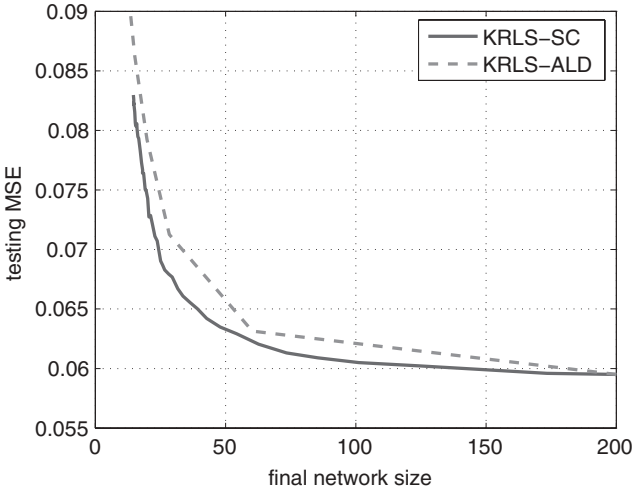


Figure 6.5. Comparison of KRLS-SC and KRLS-ALD on redundancy removal in nonlinear regression.

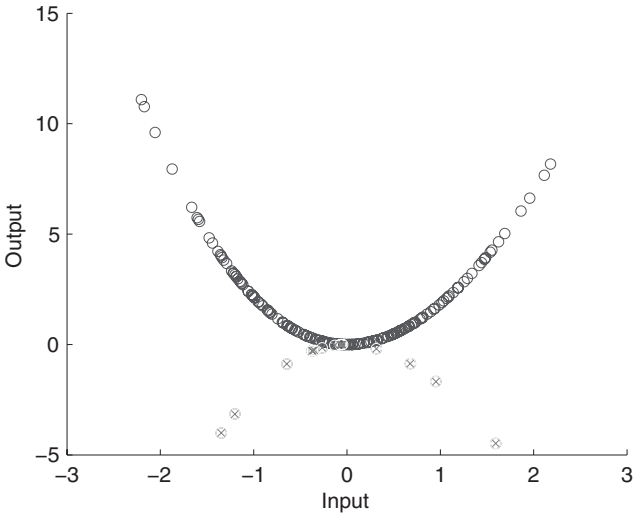


Figure 6.6. Training data with outliers (cross with gray circle) in nonlinear regression.

$T_2 = -3.37$ in ALD based on the result of the second simulation. There are actually 12 effective outliers (cross with gray circle) as shown in Figure 6.6 because another three points are close to the origin. KRLS-SC correctly detects all the outliers as shown in Figure 6.7, whereas the outliers seriously compromise the performance of KRLS-ALD as shown in Figure 6.8. This example clearly demonstrates the ability of SC to detect and reject outliers.

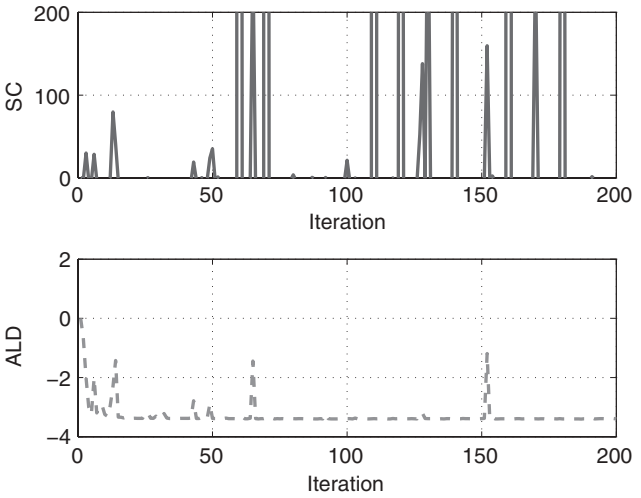


Figure 6.7. Comparison of surprise measure in KRLS-SC and ALD measure in KRLS-ALD in nonlinear regression with outliers.

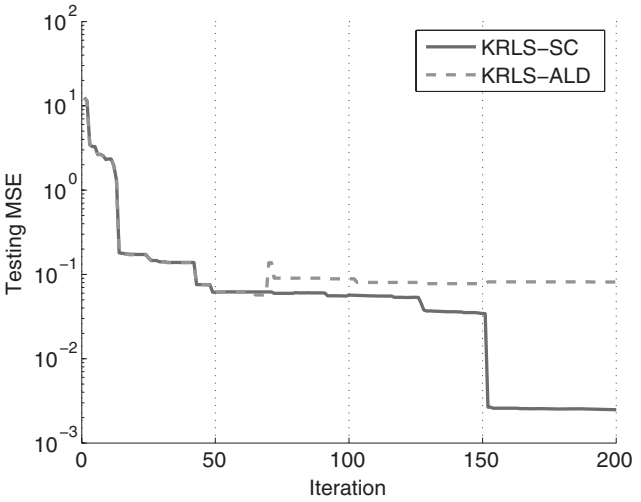


Figure 6.8. Learning curves of KRLS-SC and KRLS-ALD in nonlinear regression with outliers.

6.7.2 Surprise Criterion Applied to Mackey–Glass (MG) Time-Series Prediction

We use the MG chaotic time series again as a benchmark data set. The problem setting for short-term prediction is as follows: The previous seven points $\mathbf{u}(i) = [x(i-7), x(i-6), \ldots, x(i-1)]^T$ are used to predict the present

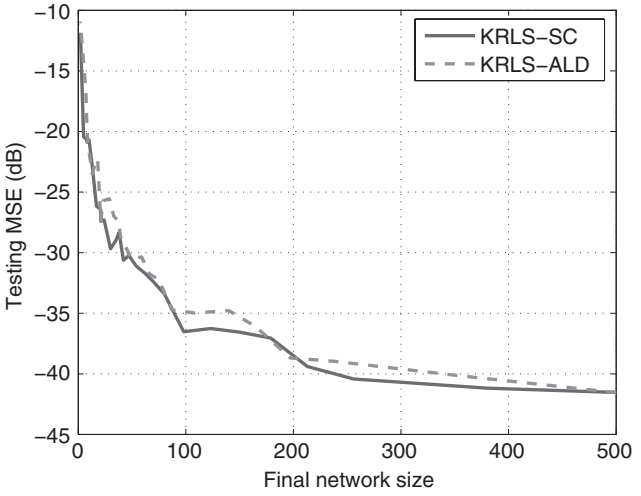


Figure 6.9. Final network size versus testing MSE of KRLS-SC and KRLS-ALD by varying T_2 in Mackey-Glass time-series prediction.

one $x(i)$. A segment of 500 samples is used as the training data and another 100 points as the test data (in the testing phase, the filter is fixed).

Part 1: First we compare the performance of KRLS-SC with KRLS-ALD. A Gaussian kernel with kernel parameter $a = 1$ is chosen. A large T_1 is used to disable the abnormality detection. We test both algorithms with 30 different T_2. The result is illustrated in Figure 6.9. It is observed that the overall performance of KRLS-SC is better than KRLS-ALD, although it is comparable. The regularization parameter $\lambda = 0.001$ is selected by cross-validation in both algorithms and memoryless uniform input distribution is assumed in the computation of surprise.

Part 2: Second, we compare the performance of a linear filter trained with LMS, novelty criterion kernel least mean square (KLMS-NC), KLMS-SC, resource-allocating network (RAN), and KRLS-SC. A Gaussian kernel with kernel parameter $a = 1$ is chosen for all the kernel based algorithms. One hundred Monte Carlo simulations are run with different realizations of noise. The noise is additive white Gaussian noise with zero mean and 0.004 variance. The step-size parameter for LMS is 0.01. The step-size parameter is 0.5 for KLMS-NC, and $\delta_1 = 0.1$ and $\delta_2 = 0.1$ are used in the novelty criterion. KLMS-SC uses step-size parameter 0.5, $T_2 = -1$, and $\lambda = 0.01$. For RAN, the step-size parameter is 0.05 and the tolerance for prediction error is 0.05. The distance resolution parameters are $\delta_{\max} = 0.5$, $\delta_{\min} = 0.05$, and $\tau_{\text{ran}} = 45$. The overlap factor is 0.87.[2] KRLS-SC uses $T_2 = -1$ and $\lambda = 0.01$. The parameters are set by cross-validation. Figure 6.10 is the ensemble learning curves for LMS, KLMS-NC, KLMS-SC, RAN, and KRLS-SC, respectively. Performance of RAN, KLMS-NC, and KLMS-SC is comparable and KRLS-SC outperforms

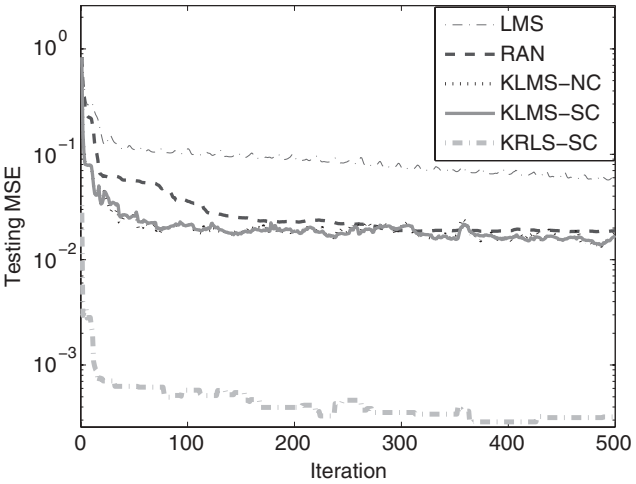


Figure 6.10. Learning curves of LMS, RAN, KLMS-NC, KLMS-SC, and KRLS-SC in Mackey-Glass time-series prediction.

Table 6.1. Comparison of network sizes of RAN, KLMS-NC, KLMS-SC, and KRLS-SC.

Algorithm	Network size
RAN	361 ± 11
KLMS-NC	201 ± 11
KLMS-SC	109 ± 8
KRLS-SC	70 ± 9

all significantly. The network sizes are listed in Table 6.1. It can be observed that KLMS-SC has a much smaller network size than KLMS-NC and RAN, which shows the superiority of the surprise criterion over the heuristic novelty criterion. In addition, the surprise criterion is simpler than the novelty criterion in the sense that it needs one threshold (only T_2; T_1 is used to remove outliers) to determine redundancy, whereas the novelty criterion requires two.

6.7.3 Surprise Criterion Applied to CO_2 Concentration Forecasting

The data consist of monthly average atmospheric CO_2 concentrations (in parts per million by volume ppmv) collected at Mauna Loa Observatory, Hawaii, between 1958 and 2008 with 603 total observations [Tans, 2008]. The first 423 points are used for training and the last 180 points are for testing. The data are shown in Figure 6.11. We try to model the CO_2 concentration as a function

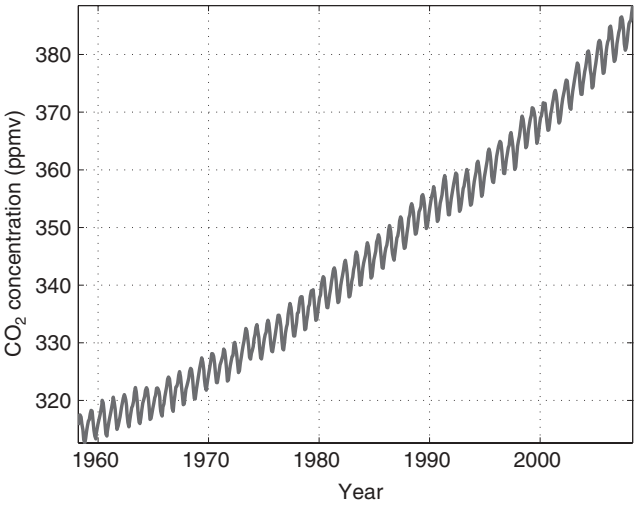


Figure 6.11. CO_2 concentration trend from year 1958 to year 2008.

of time. Several features are immediately apparent: a long term rising trend, a pronounced seasonal variation, and some smaller irregularities. The problem of kernel design for this specific task is discussed thoroughly in [Rasmussen and Williams, 2006]. We use the same kernel in this example.

A Gaussian kernel is used to model the long term smooth rising trend

$$\kappa_1(\mathbf{u}, \mathbf{u}') = \theta_1^2 \exp\left(-\frac{(\mathbf{u}-\mathbf{u}')^2}{2\theta_2^2}\right) \tag{6.26}$$

where θ_1 and θ_2 are two hyperparameters controlling the amplitude and the kernel size. A periodic kernel is used with a period of 1 year to model the seasonal effect.

$$\theta_3^2 \exp\left(-\frac{2\sin^2(\pi(\mathbf{u}-\mathbf{u}'))}{\theta_5^2}\right)$$

Because the seasonal effect may not be exactly periodic, a Gaussian kernel is multiplied to allow a decay away from exact periodicity

$$\kappa_2(\mathbf{u}, \mathbf{u}') = \theta_3^2 \exp\left(-\frac{(\mathbf{u}-\mathbf{u}')^2}{2\theta_4^2} - \frac{2\sin^2(\pi(\mathbf{u}-\mathbf{u}'))}{\theta_5^2}\right) \tag{6.27}$$

where θ_3 is the magnitude, θ_4 is the decay time for the periodic component and θ_5 is the smoothing factor of the periodic component. To model the medium term irregularities, a rational quadratic kernel is used

$$\kappa_3(\mathbf{u}, \mathbf{u}') = \theta_6^2 \left(1 + \frac{(\mathbf{u}-\mathbf{u}')^2}{2\theta_8\theta_7^2}\right)^{-\theta_8} \tag{6.28}$$

where θ_6 is the magnitude, θ_7 is the smoothing factor, and θ_8 is the shape parameter. Finally, the noise is modeled by the sum of a Gaussian kernel and a delta function

$$\kappa_4(\mathbf{u}, \mathbf{u}') = \theta_9^2 \exp\left(-\frac{-(\mathbf{u}-\mathbf{u}')^2}{2\theta_{10}^2}\right) + \theta_{11}^2 \delta(\mathbf{u}-\mathbf{u}') \tag{6.29}$$

where θ_9 is the magnitude of the correlated noise component, θ_{10} is its smoothing factor, and θ_{11} is the magnitude of the independent noise component. The delta function is defined as

$$\delta(\mathbf{u}-\mathbf{u}') = \begin{cases} 1 & \text{if } \mathbf{u} = \mathbf{u}' \\ 0 & \text{otherwise} \end{cases} \tag{6.30}$$

The final kernel function is

$$\kappa(\mathbf{u}, \mathbf{u}') = \kappa_1(\mathbf{u}, \mathbf{u}') + \kappa_2(\mathbf{u}, \mathbf{u}') + \kappa_3(\mathbf{u}, \mathbf{u}') + \kappa_4(\mathbf{u}, \mathbf{u}') \tag{6.31}$$

The hyperparameters are determined by optimizing the marginal likelihood. Multiple random initializations are tried to avoid bad local minima.[3] The goal of this example is to test how effective KRLS-SC is to model this nonlinear time series.

Part 1: At first, we simply assume all data are learnable and calculate the surprise of every point during training. The learning curve is the MSE calculated on the testing data. Figure 6.12 shows the correspondence

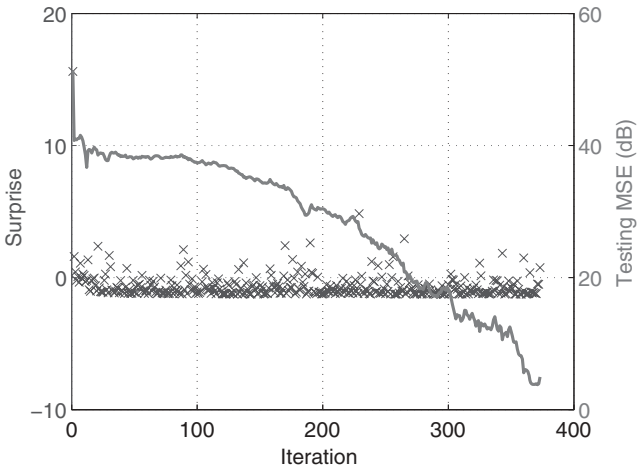


Figure 6.12. Learning curve of KRLS-SC (solid line) and surprise measure of training data (cross) along iteration assuming all data learnable in CO_2 concentration forecasting.

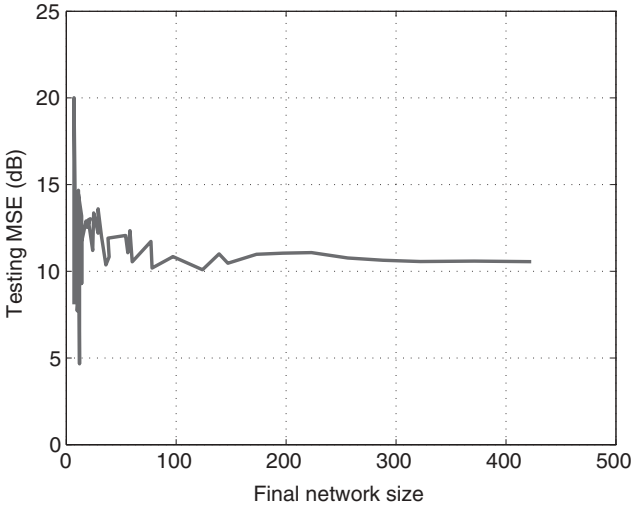


Figure 6.13. Final network size versus testing MSE of KRLS-SC by varying T_2 in CO_2 concentration forecasting.

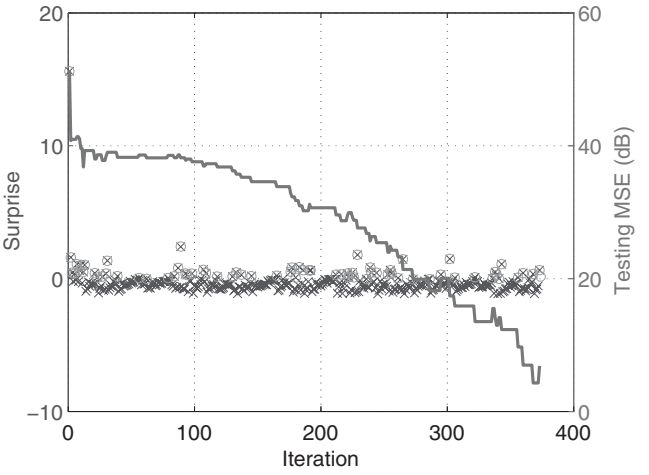


Figure 6.14. Learning curve of KRLS-SC (solid line) and surprise measure of training data (cross) along iteration with effective examples circled in CO_2 concentration forecasting.

between the additions of informative data (cross) and drops in testing MSE (solid).

Part 2: Next, we show how effective KRLS-SC is to remove redundancy. A large T_1 is used to disable the abnormality detection. Fifty different T_2 are chosen from [–1.5, 3]. The result is illustrated in Figure 6.13. The number of centers can be reduced safely from 423 to 77 with equivalent accuracy. By setting $T_2 = .03061$, we have the corresponding learning curves with the effective training data highlighted (crossed circle) in Figure 6.14. The two learning

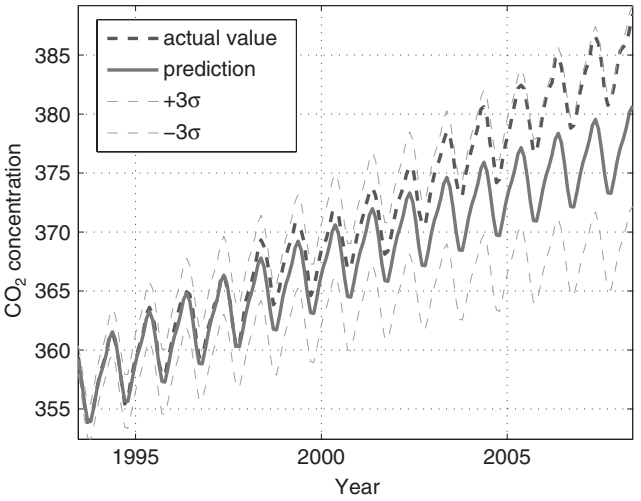


Figure 6.15. Forecasting result of KRLS-SC (T_2 = .0306) for CO_2 concentration.

curves in Figures 6.12 and 6.14 are almost the same even though the latter only uses 77 out of 423 total training data. This shows the feasibility and necessity of removing redundancy in learning. The long-term prediction result from the last training is plotted in Figure 6.15. Clearly, the prediction is accurate at the beginning but deviates in the far future. Actually, it is noted from Figure 6.15 that the increase of the CO_2 concentration seems to be accelerating at an unforeseen speed.

6.8 CONCLUSION

We present an information theoretic criterion for online active learning. Active learning is crucial in many machine learning applications, as many meaningful learning systems interact with learning environments with state models. Therefore, not all new samples encountered contain the same information to update the system state. An information measure of an example that is "transferable" to the learning system is significant. As we show theoretically and experimentally the introduced surprise measure successfully quantifies the "transferable" information content contained in a datum with respect to the learning system. The Gaussian processes theory enables an elegant and still reasonably efficient algorithm to carry on the computation in real time.

We are particularly interested in applying the surprise concept to designing sparse kernel adaptive filters. We systematically study the surprise criterion kernel adaptive filters including KRLS-SC, KLMS-SC, and KAPA-SC. We theoretically highlight the close relationship between the surprise criterion and other existing methods such as the novelty criterion and approximate linear dependency test. We show experimentally that the surprise criterion is

superior or equivalent to existing methods in the simulations of nonlinear regression, short-term chaotic time-series prediction, and long-term time-series forecasting. Moreover, the surprise criterion is more principled and elegant compared with the others.

Many more applications can benefit from this development. The interesting next question is how to apply the same technique for classification problems. The second interesting question is how to set the thresholds in the surprise criterion automatically. Any criterion requires some threshold picking. Cross-validation is usually used as a default method. Because surprise is a more meaningful quantity, it might be easier than others to choose the thresholds theoretically. Another note is about learning with abnormal data. Throwing away abnormal data may not be efficient when the learning environment is nonstationary or is subject to sudden changes. A feasible learning strategy to deal with abnormal data in a tracking mode is to make small, controlled adjustments using stochastic gradient descent. We leave all these interesting questions to future work.

ENDNOTES

1. **The Concept of Surprise**. Palm [1981] first gave a definition of surprise based on the idea of templates. Then, Itti and Baldi [2006] studied it in a Bayesian framework using the Kullback-Leibler divergence. Most recently, Ranasinghe and Shen [2008] investigated its use in robotics using symbolic schemes.
2. Please refer to Platt [1991] for the parameter settings of RAN.
3. For more details, please refer to Rasmussen and Williams [2006] pages 118–121.

EPILOGUE

In machine learning, kernel methods have already established themselves as the mathematical basis for algorithmic implementation of nonlinearly separable pattern classifiers [Schölkopf and Smola, 2002, Herbrich, 2002, Shawe-Taylor and Cristianini, 2004]. In this book, we have expanded the mathematical utility of kernel methods for algorithmic implementations of a new class of nonlinear adaptive filters, which have been named as in the following sections.

KERNEL ADAPTIVE FILTERS

The underpinnings of this new class of nonlinear filters embody the following:

- The use of Mercer kernels for nonlinear mapping of the input (data) space into a hidden (feature) space of high dimensionality.
- The use of linear adaptive filters for accommodating the adaptive requirement.

In so doing, for the first time, the design of nonlinear adaptive filters for small-scale, online adaptive signal-processing applications, such as prediction, equalization, control, modeling, and system identification, can now build on the two well-established families of linear adaptive filters.

- The least-mean-square (LMS) family that is known for its simplicity, computational efficiency, and the capability to deliver an effective and robust performance.
- The recursive least-squares (RLS) family that builds on Kalman filter theory, hence, the ability to directly exploit second-order information and thereby provide relatively fast rate of convergence.

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

Moreover, the underlying theory of kernel adaptive filters presented herein is not only unified but also elegant, elegant in that estimation of the parameters needed to implement the adaptive filters follow from one and the same algorithm, depending on the type of linear adaptive filter adopted for the design.

The impact of the book on the literature will be on the following two communities:

1. The machine-learning community is introduced to the adaptive signal-processing power of classic linear filter theory, which is exemplified by the LMS and RLS algorithms.
2. The adaptive signal-processing community are introduced to the nonlinear transformational power of Mercer kernels.

In the context of the first point, it is apropos that we close this book by reciting the title of the doctoral thesis by Rifkin [2002]:

> Everything old is new again: A fresh look at historical approaches in machine learning.

In his thesis, Rifkin argues for attention be given to the role of least-squares estimation in machine learning. In a dual way, in this book we have argued for the role of kernel methods in adaptive signal-processing, thereby hoping to bring the two communities closer together.

APPENDIX A
MATHEMATICAL BACKGROUND

A.1 SINGULAR VALUE DECOMPOSITION

The singular value decomposition (SVD) is a powerful factorization tool that is useful for both analytical and numerical purposes. Suppose $\mathbf{A}$ is an $m \times n$ real matrix. Without loss of generality, we assume $m \geq n$. Then there exists an $m \times m$ orthogonal matrix $\mathbf{U}$ ($\mathbf{U}\mathbf{U}^T = \mathbf{I}$), an $n \times n$ orthogonal matrix $\mathbf{V}$ ($\mathbf{V}\mathbf{V}^T = \mathbf{I}$), and an $n \times n$ diagonal matrix $\mathbf{S}$ with nonnegative entries on the diagonal such that

$$\mathbf{A} = \mathbf{U}\begin{bmatrix}\mathbf{S}\\ \mathbf{0}\end{bmatrix}\mathbf{V}^T \tag{A.1}$$

where $\mathbf{0}$ is an $(m-n)\times n$ zero matrix. Such a factorization is called the singular value decomposition of $\mathbf{A}$. The nonzero diagonal entries of $\mathbf{S}$ are called the *singular values* of $\mathbf{A}$ and are usually ordered in decreasing order,

$$\mathbf{S} = \text{diag}(s_1, \ldots, s_r, 0, \ldots, 0)$$

with

$$s_1 \geq \ldots \geq s_r > 0$$

If $\mathbf{S}$ has r nonzero diagonal entries, then $\mathbf{A}$ has rank r. The columns of $\mathbf{U}$ and $\mathbf{V}$ are called the left- and right-singular vectors of $\mathbf{A}$, respectively. And they satisfy

$$\mathbf{A}\mathbf{V}_j = s_j\mathbf{U}_j, \quad j = 1, \ldots, r \tag{A.2}$$

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

and

$$\mathbf{A}^T\mathbf{U}_j = s_j\mathbf{V}_j, \quad j = 1, \ldots, r \tag{A.3}$$

where $\mathbf{U}_j$ and $\mathbf{V}_j$ denotes the jth column of $\mathbf{U}$ and $\mathbf{V}$, respectively.

A.1.1 Pseudo Inverse

The pseudo inverse of a matrix is a generalization of the concept of inverses for square invertible matrices. It is defined for matrices that need not be square or even invertible.

Suppose $\mathbf{A}$ is an $m \times n$ $(m \geq n)$ real matrix that has its singular value decomposition as

$$\mathbf{A} = \mathbf{U}\begin{bmatrix}\mathbf{S}\\ \mathbf{0}\end{bmatrix}\mathbf{V}^T$$

Then, its pseudo inverse can be determined as

$$\mathbf{A}^\dagger = \mathbf{V}\begin{bmatrix}\mathbf{S}^\dagger & \mathbf{0}\end{bmatrix}\mathbf{U}^T \tag{A.4}$$

where

$$\mathbf{S}^\dagger = \text{diag}\left\{s_1^{-1}, s_2^{-1}, \ldots s_r^{-1}, 0, \ldots, 0\right\} \tag{A.5}$$

It is easy to verify that the pseudo inverse has the following properties:

$$\begin{aligned}\mathbf{A}\mathbf{A}^\dagger\mathbf{A} &= \mathbf{A}\\ \mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger &= \mathbf{A}^\dagger\\ \left(\mathbf{A}\mathbf{A}^\dagger\right)^T &= \mathbf{A}\mathbf{A}^\dagger\\ \left(\mathbf{A}^\dagger\mathbf{A}\right)^T &= \mathbf{A}^\dagger\mathbf{A}\end{aligned} \tag{A.6}$$

A.1.2 Minimum Norm Solution

Consider a least-squares problem

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{A}\mathbf{w}\|^2 \tag{A.7}$$

with possibly an infinite number of solutions. Then the minimum norm solution is unique and is given by

$$\mathbf{w}_{PI} = \mathbf{A}^\dagger\mathbf{y} \tag{A.8}$$

A.2 POSITIVE-DEFINITE MATRIX

An $n \times n$ real symmetric matrix $\mathbf{A}$ is *positive definite* if $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for all nonzero vectors $\mathbf{x}$. The following properties are equivalent to $\mathbf{A}$ being positive definite:

1. All eigenvalues ς_j of $\mathbf{A}$ are positive.
2. The bilinear form $\mathbf{x}^T\mathbf{A}\mathbf{y}$ of $\mathbf{x}$ and $\mathbf{y}$ defines an inner product on $\mathbb{R}^n$.
3. $\mathbf{A}$ is the Gram matrix of some collection of linearly independent vectors. In other words, an $n \times m$ ($m \geq n$) full-rank matrix $\mathbf{M}$ exists such that

$$\mathbf{A} = \mathbf{M}\mathbf{M}^T$$

4. A unique lower triangular matrix $\mathbf{L}$ exists with positive diagonal entries such that

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

This factorization is called *Cholesky decomposition*.

A matrix $\mathbf{A}$ is *positive semidefinite* if $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$ for all nonzero vectors $\mathbf{x}$. $\mathbf{A}$ is the Gram matrix of some collection of vectors that may not be linearly independent. In other words, an $n \times m$ matrix $\mathbf{M}$ exists such that

$$\mathbf{A} = \mathbf{M}\mathbf{M}^T$$

A.3 EIGENVALUE DECOMPOSITION

Given an $n \times n$ matrix $\mathbf{A}$, if a nonzero vector $\mathbf{q}$ satisfies the following condition:

$$\mathbf{A}\mathbf{q} = \varsigma\mathbf{q} \tag{A.9}$$

then we call $\mathbf{q}$ is an *eigenvector* of $\mathbf{A}$ with corresponding *eigenvalue* ς. Because

$$(\mathbf{A} - \varsigma\mathbf{I})\mathbf{q} = \mathbf{0} \tag{A.10}$$

has a nonzero solution, the matrix $(\mathbf{A} - \varsigma\mathbf{I})$ must be singular; in other words, the determinant of this matrix must be zero

$$|\mathbf{A} - \varsigma\mathbf{I}| = 0 \tag{A.11}$$

This is the most common way to compute the eigenvalues and eigenvectors of a given matrix.

If $\mathbf{A}$ has n linearly independent eigenvectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$, then $\mathbf{A}$ has the following eigenvalue decomposition

$$\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}^{-1}$$

where $\mathbf{Q}$ is an $n \times n$ matrix whose ith column is $\mathbf{q}_i$ and Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues

$$\Lambda = diag\{\varsigma_1, \ldots, \varsigma_n\}$$

Furthermore, we have the following identities:

$$\mathrm{tr}[\mathbf{A}] = \mathrm{tr}[\Lambda] = \sum_{j=1}^{n} \varsigma_j$$

$$|\mathbf{A}| = |\Lambda| = \prod_{j=1}^{n} \varsigma_j$$

Assume $\mathbf{A}$ has a form of

$$\mathbf{A} = \mathbf{M}^T\mathbf{M}$$

with arbitrary matrix $\mathbf{M}$. $\mathbf{M}$ has the singular value decomposition

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^T$$

Then, it is easy to show that $\mathbf{A}$ has the following eigenvalue decomposition:

$$\mathbf{A} = \mathbf{V}\left(\Sigma^T\Sigma\right)\mathbf{V}^T$$

The columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{A}$ and the square of the singular values of $\mathbf{M}$ are the eigenvalues of $\mathbf{A}$.

However, if another matrix has a form of

$$\mathbf{B} = \mathbf{M}\mathbf{M}^T$$

then

$$\mathbf{B} = \mathbf{U}\left(\Sigma\Sigma^T\right)\mathbf{U}^T$$

The columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{B}$ and the square of the singular values of $\mathbf{M}$ are the eigenvalues of $\mathbf{B}$. $\mathbf{A}$ and $\mathbf{B}$ share the same nonzero eigenvalues.

A.4 SCHUR COMPLEMENT

Consider an invertible block matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

The Schur complement of $\mathbf{A}$ in $\mathbf{M}$ is denoted by $\mathbf{S}_\mathbf{A}$ and is defined as

$$\mathbf{S}_\mathbf{A} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B} \tag{A.12}$$

Likewise, the Schur complement of $\mathbf{D}$ in $\mathbf{M}$ is denoted by $\mathbf{S}_\mathbf{D}$ and is defined as

$$\mathbf{S}_\mathbf{D} = \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C} \tag{A.13}$$

Using these Schur complements, it is easy to verify that the block matrix $\mathbf{M}$ can be factored in either of the following forms:

$$\begin{aligned} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} &= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_\mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{I} & \mathbf{B}\mathbf{D}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{S}_\mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{bmatrix} \end{aligned} \tag{A.14}$$

A.5 BLOCK MATRIX INVERSE

Let an invertible block matrix be

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

then by equation (A.14), its inverse can be computed as

$$\begin{aligned} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} &= \begin{bmatrix} \mathbf{I} & -\mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_\mathbf{A}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\mathbf{S}_\mathbf{A}^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{S}_\mathbf{A}^{-1} \\ -\mathbf{S}_\mathbf{A}^{-1}\mathbf{C}\mathbf{A}^{-1} & \mathbf{S}_\mathbf{A}^{-1} \end{bmatrix} \end{aligned} \tag{A.15}$$

or in another way

$$\begin{aligned}\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1} &= \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ -\mathbf{D}^{-1}\mathbf{C} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{S}_{\mathbf{D}}^{-1} & \mathbf{0}\\ \mathbf{0} & \mathbf{D}^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{I} & -\mathbf{B}\mathbf{D}^{-1}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{S}_{\mathbf{D}}^{-1} & -\mathbf{S}_{\mathbf{D}}^{-1}\mathbf{B}\mathbf{D}^{-1}\\ -\mathbf{D}^{-1}\mathbf{C}\mathbf{S}_{\mathbf{D}}^{-1} & \mathbf{D}^{-1}+\mathbf{D}^{-1}\mathbf{C}\mathbf{S}_{\mathbf{D}}^{-1}\mathbf{B}\mathbf{D}^{-1}\end{bmatrix}\end{aligned} \tag{A.16}$$

By equating these two results, we obtain the following identities:

$$\begin{aligned}\mathbf{S}_{\mathbf{D}}^{-1} &= \mathbf{A}^{-1}+\mathbf{A}^{-1}\mathbf{B}\mathbf{S}_{\mathbf{A}}^{-1}\mathbf{C}\mathbf{A}^{-1}\\ \mathbf{S}_{\mathbf{A}}^{-1} &= \mathbf{D}^{-1}+\mathbf{D}^{-1}\mathbf{C}\mathbf{S}_{\mathbf{D}}^{-1}\mathbf{B}\mathbf{D}^{-1}\end{aligned} \tag{A.17}$$

$$\begin{aligned}\mathbf{S}_{\mathbf{A}}^{-1}\mathbf{C}\mathbf{A}^{-1} &= \mathbf{D}^{-1}\mathbf{C}\mathbf{S}_{\mathbf{D}}^{-1}\\ \mathbf{A}^{-1}\mathbf{B}\mathbf{S}_{\mathbf{A}}^{-1} &= \mathbf{S}_{\mathbf{D}}^{-1}\mathbf{B}\mathbf{D}^{-1}\end{aligned} \tag{A.18}$$

$\mathbf{S}_{\mathbf{A}}$ and $\mathbf{S}_{\mathbf{D}}$ are the Schur complements of $\mathbf{A}$ and $\mathbf{D}$, respectively.

The equation

$$\mathbf{S}_{\mathbf{D}}^{-1} = \mathbf{A}^{-1}+\mathbf{A}^{-1}\mathbf{B}\mathbf{S}_{\mathbf{A}}^{-1}\mathbf{C}\mathbf{A}^{-1}$$

is the basis of the proof of the following matrix inversion lemma.

A.6 MATRIX INVERSION LEMMA

The matrix inversion lemma is also called Woodbury matrix identity, which is named after Max A. Woodbury. It states that

$$(\mathbf{A}+\mathbf{B}\mathbf{C}\mathbf{D})^{-1} = \mathbf{A}^{-1}-\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{C}^{-1}+\mathbf{D}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{D}\mathbf{A}^{-1} \tag{A.19}$$

This identity can be derived directly from the results of the block matrix inverse.

A.7 JOINT, MARGINAL, AND CONDITIONAL PROBABILITY

Given two random variables (vectors) $\mathbf{x}$ and $\mathbf{y}$, the *joint distribution* $p(\mathbf{x}, \mathbf{y})$ of $\mathbf{x}$ and $\mathbf{y}$ defines the probability of events defined in terms of both $\mathbf{x}$ and $\mathbf{y}$. The *marginal distribution* of $\mathbf{y}$ is given by

$$p(\mathbf{y}) = \int p(\mathbf{x}, \mathbf{y})\, d\mathbf{x}$$

If the joint distribution equals to the product of the marginals, i.e.,

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})\, p(\mathbf{y})$$

then the variables are said to be *independent*. Otherwise, they are *dependent*. The *conditional distribution* is defined as

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})}$$

given $p(\mathbf{x}) > 0$. If $\mathbf{x}$ and $\mathbf{y}$ are independent, then the conditional distribution is equal to the marginal. Using the definitions of $p(\mathbf{x}|\mathbf{y})$ and $p(\mathbf{y}|\mathbf{x})$, we have the *Bayes' rule*

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})} \tag{A.20}$$

A.8 NORMAL DISTRIBUTION

A random vector $\mathbf{x}$ is said to follow a *multivariate normal distribution* if its probability density function is

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{N/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{m})\right)$$

where $\mathbf{m}$ is the mean

$$\mathbf{m} = \mathbf{E}[\mathbf{x}]$$

and $\mathbf{\Sigma}$ is the covariance matrix

$$\mathbf{\Sigma} = \mathbf{E}\left[(\mathbf{x}-\mathbf{m})(\mathbf{x}-\mathbf{m})^T\right]$$

$|\mathbf{\Sigma}|$ is the determinant of $\mathbf{\Sigma}$. A multivariate normal distribution is also called *multivariate Gaussian distribution*, which is usually written in the following notation:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{m}, \mathbf{\Sigma})$$

Let $\mathbf{x}$ and $\mathbf{y}$ be jointly Gaussian distributed

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{m}_x\\ \mathbf{m}_y\end{bmatrix}, \begin{bmatrix}\mathbf{A} & \mathbf{C}\\ \mathbf{C}^T & \mathbf{B}\end{bmatrix}\right)$$

then the marginal distribution of $\mathbf{x}$ is

$$\mathbf{x} \sim \mathcal{N}(\mathbf{m}_x, \mathbf{A})$$

and the conditional distribution of $\mathbf{x}$ given $\mathbf{y}$ is

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}\left(\mathbf{m}_x + \mathbf{C}\mathbf{B}^{-1}\left(\mathbf{y} - \mathbf{m}_y\right), \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^T\right)$$

A.9 GRADIENT DESCENT

Gradient descent is also known as *steepest descent*, or the method of steepest descent. It is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

Suppose the cost is a differentiable real-valued function $J(\mathbf{x})$ and the starting point is $\mathbf{x}(0)$. Define

$$\mathbf{x}(1) = \mathbf{x}(0) - \eta\, \nabla J(\mathbf{x}(0))$$

then it can be shown that

$$J(\mathbf{x}(1)) \leq J(\mathbf{x}(0))$$

provided that the step-size parameter $\eta(>0)$ is small enough. By the Taylor expansion of $J(\mathbf{x})$ around $\mathbf{x}(0)$, we have

$$\begin{aligned} J(\mathbf{x}(1)) &= J(\mathbf{x}(0)) + \nabla J(\mathbf{x}(0))(\mathbf{x}(1) - \mathbf{x}(0)) + O(\mathbf{x}(1) - \mathbf{x}(0))^2 \\ &= J(\mathbf{x}(0)) + \nabla J(\mathbf{x}(0))[-\eta \nabla J(\mathbf{x}(0))] + O(\eta \nabla J(\mathbf{x}(0)))^2 \\ &= J(\mathbf{x}(0)) - \eta[\nabla J(\mathbf{x}(0))]^2 + \eta^2 O(\nabla J(\mathbf{x}(0)))^2 \\ &< J(\mathbf{x}(0)) \quad (\eta \text{ is small}) \end{aligned}$$

Define a sequence $\mathbf{x}(0), \mathbf{x}(1), \mathbf{x}(2), \ldots$ such that

$$\mathbf{x}(i+1) = \mathbf{x}(i) - \eta \nabla J(\mathbf{x}(i))$$

we have

$$J(\mathbf{x}(0)) > J(\mathbf{x}(1)) > J(\mathbf{x}(2)) \ldots$$

and it converges to a local minimum where $\nabla J = \mathbf{0}$.

A.10 NEWTON'S METHOD

Newton's method is a well-known algorithm for finding the roots of equations in one or more dimensions. It can also be used to find local minima. The geometric interpretation of Newton's method is that at each iteration, one

approximates the cost $J(\mathbf{x})$ by a quadratic function and then takes a step toward the minimum of that quadratic function. If $J(\mathbf{x})$ happens to be a quadratic function, then the exact extremum is found in one step. We use the Taylor expansion to approximate the cost function around $\mathbf{x}$

$$J(\mathbf{x}+\mathbf{h}) \approx J(\mathbf{x}) + \nabla_{\mathbf{x}} J(\mathbf{x})\mathbf{h} + \frac{1}{2}\mathbf{h}^T \mathbf{H}(\mathbf{x})\mathbf{h}$$

where $\nabla_{\mathbf{x}} J(\mathbf{x})$ is the gradient of $J(\mathbf{x})$ with respect to $\mathbf{x}$ and $\mathbf{H}(\mathbf{x})$ is the *Hessian matrix* defined as

$$\mathbf{H}(\mathbf{x}) = \nabla_{\mathbf{x}} \nabla_{\mathbf{x}} J(\mathbf{x})$$

To minimize the cost, we select $\mathbf{h}$ such that this quadratic function attains extremum. It is well-known that the extremum is attained at the stationary point

$$\nabla_{\mathbf{h}} J(\mathbf{x}+\mathbf{h}) = \nabla_{\mathbf{x}} J(\mathbf{x}) + \mathbf{H}(\mathbf{x})\mathbf{h} = 0$$

or equivalently

$$\mathbf{h} = -\mathbf{H}(\mathbf{x})^{-1} \nabla_{\mathbf{x}} J(\mathbf{x})$$

By this observation, we have the following iterative scheme:

$$\mathbf{x}(i+1) = \mathbf{x}(i) - \eta \mathbf{H}(\mathbf{x}(i))^{-1} \nabla_{\mathbf{x}} J(\mathbf{x}(i))$$

The computation of the inverse of the Hessian matrix is expensive, many *quasi-Newton methods* exist to solve the problem approximately.

APPENDIX B

APPROXIMATE LINEAR DEPENDENCY AND SYSTEM STABILITY

At iteration i of kernel recursive least-squares algorithm (KRLS), we solve the following linear system for $\mathbf{a}(i)$:

$$[\mathbf{G}(i)+\lambda\mathbf{I}]\mathbf{a}(i)=\mathbf{d}(i) \tag{B.1}$$

At the next iteration, a new data example, $\{\mathbf{u}(i+1), d(i+1)\}$ will become available and the learning system will need to be updated accordingly. Obviously, $\mathbf{a}(i+1)$ satisfies the following linear equation:

$$[\mathbf{G}(i+1)+\lambda\mathbf{I}]\mathbf{a}(i+1)=\mathbf{d}(i+1) \tag{B.2}$$

where

$$\mathbf{G}(i+1)=\begin{bmatrix}\mathbf{G}(i) & \mathbf{h}(i+1)\\ \mathbf{h}(i+1)^T & \kappa(\mathbf{u}(i+1),\mathbf{u}(i+1))\end{bmatrix} \tag{B.3}$$

$$\mathbf{h}(i+1)=[\kappa(\mathbf{u}(i+1),\mathbf{u}(1)),\ldots,\kappa(\mathbf{u}(i+1),\mathbf{u}(i))]^T \tag{B.4}$$

$$\mathbf{d}(i+1)=[d(1),\ldots,d(i+1)]^T \tag{B.5}$$

$\mathbf{G}(i+1)$ is the Gram matrix at time $i+1$, and apparently it has a growing structure based on $\mathbf{G}(i)$. Denoting

$$\mathbf{Q}(i)=[\mathbf{G}(i)+\lambda\mathbf{I}]^{-1}$$

and using the Woodbury matrix identity, we have

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

$$\mathbf{Q}(i+1)=\begin{bmatrix}\mathbf{Q}(i)+\mathbf{z}(i+1)\mathbf{z}(i+1)^T r(i+1)^{-1} & -\mathbf{z}(i+1)r(i+1)^{-1}\\ -\mathbf{z}(i+1)^T r(i+1)^{-1} & r(i+1)^{-1}\end{bmatrix} \tag{B.6}$$

where

$$\begin{aligned}\mathbf{z}(i+1)&=\mathbf{Q}(i)\mathbf{h}(i+1)\\ r(i+1)&=\lambda+\kappa(\mathbf{u}(i+1),\mathbf{u}(i+1))-\mathbf{z}(i+1)^T\mathbf{h}(i+1)\end{aligned} \tag{B.7}$$

$r(i+1)$ is the Schur complement of $\mathbf{G}(i)+\lambda\mathbf{I}$ w.r.t. $\mathbf{G}(i+1)+\lambda\mathbf{I}$. As we observe from equation (B.6) that when $r(i+1)\to 0$, some terms in $\mathbf{Q}(i+1)$ approach infinity, and the solution could be out of control. An ill-conditioned system matrix is undesirable in any circumstance.

We are interested in the following question: If the system is stable at time i, given the new data $\mathbf{u}(i+1)$, what can we say about the system at time $i+1$? This is particularly important in terms of active data selection. If the inclusion of new data renders instability, it should not be included into the system anyway. Comparing with the regularization methodology, this can be regarded as a "preemptive" approach for the learning system to stay stable. Because the minimum eigenvalue measures the distance of the matrix from being singular, a mathematical restatement of the problem is: If the minimum eigenvalue of $\mathbf{G}(i)$ is $\varsigma_{\min}(i)$, then what would be the minimum eigenvalue $\varsigma_{\min}(i+1)$ of $\mathbf{G}(i+1)$?

We will show that $\varsigma_{\min}(i+1)$ is almost proportional to the smaller value of $\varsigma_{\min}(i)$ and $r(i+1)$. For simplicity, we assume $\lambda=0$ in the following treatment. All derivations and results hold with a nonzero λ [just redefine $\mathbf{G}(i)$ as $\mathbf{G}(i)+\lambda\mathbf{I}$].

First, we prove an important lemma on the Schur complement $r(i+1)$.

Lemma B.1. *$\mathbf{G}(i)$ is invertible. $r(i+1)$ is the Schur complement of $\mathbf{G}(i)$ w.r.t. $\mathbf{G}(i+1)$ defined in equation* (B.7). *For any real number* v, *the following equation holds:*

$$\langle r(i+1)v, v\rangle=\inf_{\mathbf{w}}\left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}\\ v\end{bmatrix},\begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}\right\rangle \tag{B.8}$$

Proof. Let $\mathbf{w}_v=-\mathbf{G}(i)^{-1}\mathbf{h}(i+1)v$. Then, equation using (B.3), we have

$$\mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}=\begin{bmatrix}0\\ r(i+1)v\end{bmatrix}$$

Hence,

$$\langle r(i+1)v, v\rangle=\left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix},\begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}\right\rangle \tag{B.9}$$

Therefore,

$$\langle r(i+1)v, v\rangle \geq \inf_{\mathbf{w}} \left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}\right\rangle \tag{B.10}$$

To prove the reverse inequality (≤), observe that

$$\left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}\right\rangle = \langle r(i+1)v, v\rangle = \left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}\right\rangle \tag{B.11}$$

for any $\mathbf{w}$. Applying Cauchy-Schwarz inequality in the inner product generated by $\mathbf{G}(i+1)$, we have

$$\left[\left\langle \begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}\right\rangle \mathbf{G}(i+1)\right]^2 \leq \left\langle \begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}\right\rangle \mathbf{G}(i+1) \left\langle \begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}\right\rangle \mathbf{G}(i+1) \tag{B.12}$$

By using equation (B.11) and canceling a common factor, we have that

$$\langle r(i+1)v, v\rangle = \left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}_v\\ v\end{bmatrix}\right\rangle \leq \left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}\right\rangle \tag{B.13}$$

for all $\mathbf{w}$. Taking the infimum over all $\mathbf{w}$, we find that the reverse inequality also holds. □

Also by the Woodbury matrix identity, it is easy to show that

$$0 \leq r(i+1) \leq \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) \tag{B.14}$$

In the case $\lambda \neq 0$, we have

$$\lambda \leq r(i+1) \leq \lambda + \kappa(\mathbf{u}(i+1), \mathbf{u}(i+1)) \tag{B.15}$$

in general.

We have the following upper bound of $\varsigma_{\min}(i+1)$ based on $\varsigma_{\min}(i)$ and $r(i+1)$.

Theorem B.2.

$$\varsigma_{\min}(i+1) \leq \min\{\varsigma_{\min}(i), r(i+1)\} \tag{B.16}$$

Proof. First of all, by the interlacing theorem [Golub and Loan, 1996], we have

$$\varsigma_{\min}(i+1) \leq \varsigma_{\min}(i) \tag{B.17}$$

Furthermore, we have

$$\inf_{\mathbf{w}} \left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}\\ v\end{bmatrix} \right\rangle \geq \inf_{\mathbf{x}:\|\mathbf{x}\|^2 \geq v^2} \langle \mathbf{G}(i+1)\mathbf{x}, \mathbf{x}\rangle \tag{B.18}$$

$$= \inf_{\mathbf{x}:\|\mathbf{x}\|^2 = v^2} \langle \mathbf{G}(i+1)\mathbf{x}, \mathbf{x}\rangle \tag{B.19}$$

$$= \varsigma_{\min}(i+1)v^2 \tag{B.20}$$

which is true for any v.

By using Lemma B.1, we have

$$r(i+1) \geq \varsigma_{\min}(i+1) \tag{B.21}$$

Therefore, by combining equations (B.17) and (B.21), we complete the proof. □

This theorem actually corroborates our observations in the previous section that a small $r(i + 1)$ will lead $\mathbf{G}(i + 1)$ to singularity, and it shows that an unstable system cannot be "salvaged" by adding new data. This fact directly encourages us to discard the data point $\mathbf{u}(i + 1)$ in the case of $r(i + 1)$ being too small.

Discarding a "bad" data point with a small $r(i + 1)$ certainly helps, but this theorem does not guarantee a stable system at time $i + 1$ by adding a data point with a "not-too-small" $r(i + 1)$. In other words, the upper bound provides a necessary condition to have a stable system, but a sufficient condition is more important.

Now we prove the following theorem, which establishes a lower bound of $\varsigma_{\min}(i + 1)$ based on $\varsigma_{\min}(i)$ and $r(i + 1)$.

Theorem B.3. *For simplicity, denote* $g := \kappa(\mathbf{u}(i + 1), \mathbf{u}(i + 1))$, $r := r(i + 1)$ *and* $\mathbf{h} := \mathbf{h}(i + 1)$.

$$\varsigma_{\min}(i+1) \geq \frac{2\varsigma_{\min}(i)r}{(g+r+\varsigma_{\min}(i)) + \sqrt{(g+r+\varsigma_{\min}(i))^2 - 8\varsigma_{\min}(i)r}} \tag{B.22}$$

Proof. The proof is complicated and we break it into three parts.

First, by the property of the smallest eigenvalue [Golub and Loan, 1996]

$$\inf_{\mathbf{w},v,\|\mathbf{w}\|^2+\|v\|^2=1} \left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}\\ v\end{bmatrix}, \begin{bmatrix}\mathbf{w}\\ v\end{bmatrix} \right\rangle = \varsigma_{\min}(i+1) \tag{B.23}$$

Denoting the eigenvector of $\mathbf{G}(i + 1)$ w.r.t. $\varsigma_{\min}(i + 1)$ is $[\mathbf{z}, v_1]^T$ with $\|[\mathbf{z}, v_1]^T\| = 1$, we have

$$\begin{aligned}\varsigma_{\min}(i+1) &= \left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{z}\\ v_1\end{bmatrix}, \begin{bmatrix}\mathbf{z}\\ v_1\end{bmatrix}\right\rangle \\ &\geq \inf_{\mathbf{w}} \left\langle \mathbf{G}(i+1)\begin{bmatrix}\mathbf{w}\\ v_1\end{bmatrix}, \begin{bmatrix}\mathbf{w}\\ v_1\end{bmatrix}\right\rangle \\ &= rv_1^2 \quad \text{(by Lemma B.1)}\end{aligned} \tag{B.24}$$

Hence, we have a lower bound for $\varsigma_{\min}(i + 1)$,

$$\varsigma_{\min}(i+1) \geq rv_1^2 \tag{B.25}$$

However if $|v_1|$ is too small, then this lower bound is trivial.

Second, v_1 is not arbitrary and is interrelated with $\mathbf{z}$ and $\varsigma_{\min}(i + 1)$ by the following equation:

$$\varsigma_{\min}(i+1)\begin{bmatrix}\mathbf{z}\\ v_1\end{bmatrix} = \mathbf{G}(i+1)\begin{bmatrix}\mathbf{z}\\ v_1\end{bmatrix}$$

or equivalently,

$$\mathbf{G}(i)\mathbf{z} + \mathbf{h}v_1 = \varsigma_{\min}(i+1)\mathbf{z} \tag{B.26}$$

$$\mathbf{h}^T\mathbf{z} + gv_1 = \varsigma_{\min}(i+1)v_1 \tag{B.27}$$

If $v_1 = 0$, we have

$$\mathbf{G}(i)\mathbf{z} = \varsigma_{\min}(i+1)\mathbf{z} \tag{B.28}$$

$$\mathbf{h}^T\mathbf{z} = 0 \tag{B.29}$$

Because $\mathbf{z}$ is nonzero in this case, $\varsigma_{\min}(i + 1)$ is the eigenvalue of $\mathbf{G}(i)$ and by definition $\varsigma_{\min}(i + 1) \geq \varsigma_{\min}(i)$. That is, the lower bound equation (B.25) is loose when v_1 is small.

The idea is to use these constraints in equations (B.26) and (B.27) to get a better bound. By multiplying $\mathbf{z}^T$ on equation (B.26), we have

$$\mathbf{z}^T\mathbf{h}v_1 = \mathbf{z}^T\varsigma_{\min}(i+1)\mathbf{z} - \mathbf{z}^T\mathbf{G}(i)\mathbf{z} \tag{B.30}$$

Because $\mathbf{z}^T\mathbf{h} = \mathbf{h}^T\mathbf{z}$, by substituting equation (B.27) into equation (B.30), we have

$$(g - \varsigma_{\min}(i+1))v_1^2 = -\varsigma_{\min}(i+1)\|\mathbf{z}\|^2 + \mathbf{z}^T\mathbf{G}(i)\mathbf{z} \tag{B.31}$$

Recall that $\|\mathbf{z}\|^2 + v_1^2 = 1$. If $v_1^2 = 1$ and $\|\mathbf{z}\|^2 = 0$, we have $\varsigma_{\min}(i + 1) = g$, which is greater than $r(i + 1)$ and consistent with (B.25). Now, if $\|\mathbf{z}\|^2 > 0$, we have

$$\frac{\mathbf{z}^T\mathbf{G}(i)\mathbf{z}}{\|\mathbf{z}\|^2} = (g - \varsigma_{\min}(i+1))\frac{v_1^2}{\|\mathbf{z}\|^2} + \varsigma_{\min}(i+1) \tag{B.32}$$

Therefore, it follows that

$$(g - \varsigma_{\min}(i+1))\frac{v_1^2}{\|\mathbf{z}\|^2} + \varsigma_{\min}(i+1) = \frac{\mathbf{z}^T\mathbf{G}(i)\mathbf{z}}{\|\mathbf{z}\|^2} \geq \varsigma_{\min}(i) \tag{B.33}$$

After some rearrangement, we have the second lower bound

$$\left(1 - \frac{v_1^2}{\|\mathbf{z}\|^2}\right)\varsigma_{\min}(i+1) \geq -g\frac{v_1^2}{\|\mathbf{z}\|^2} + \varsigma_{\min}(i) \tag{B.34}$$

The third part is to combine the two lower bounds just derived above. Notice that on the one hand, if v_1^2 is close to 1, then equation (B.25) is tighter, on the other hand, if v_1^2 is close to 0, then equation (B.34) is much tighter. It suffices to know that $\varsigma_{\min}(i+1)$ cannot be arbitrarily small. The lowest value on the lower bound is achieved at the cross point of the two, i.e.,

$$\left(1 - \frac{v_1^2}{\|\mathbf{z}\|^2}\right)rv_1^2 = -g\frac{v_1^2}{\|\mathbf{z}\|^2} + \varsigma_{\min}(i) \tag{B.35}$$

Solving this equation, we have

$$v_1^2 = \frac{(g + r + \varsigma_{\min}(i)) \pm \sqrt{(g + r + \varsigma_{\min}(i))^2 - 8\varsigma_{\min}(i)r}}{4r} \tag{B.36}$$

Notice that $r = g - \mathbf{h}^T\mathbf{G}(i)^{-1}\mathbf{h} \leq g$, so

$$\begin{aligned}(g + r + \varsigma_{\min}(i))^2 - 8\varsigma_{\min}(i)r &\geq (2r + \varsigma_{\min}(i))^2 - 8\varsigma_{\min}(i)r \\ &= (2r - \varsigma_{\min}(i))^2 \\ &\geq 0\end{aligned}$$

Picking the smaller solution and using equation (B.25), we have

$$\varsigma_{\min}(i+1) \geq \frac{(g + r + \varsigma_{\min}(i)) - \sqrt{(g + r + \varsigma_{\min}(i))^2 - 8\varsigma_{\min}(i)r}}{4} \tag{B.37}$$

which holds in all situations. Notice that

$$\frac{(g+r+\varsigma_{\min}(i))-\sqrt{(g+r+\varsigma_{\min}(i))^2-8\varsigma_{\min}(i)r}}{4} = \frac{2\varsigma_{\min}(i)r}{(g+r+\varsigma_{\min}(i))+\sqrt{(g+r+\varsigma_{\min}(i))^2-8\varsigma_{\min}(i)r}} \tag{B.38}$$

This completes the proof. □

A relaxed and simpler lower bound is

$$\varsigma_{\min}(i+1) \geq \frac{\varsigma_{\min}(i)r}{g+r+\varsigma_{\min}(i)} \tag{B.39}$$

To conclude, "not-too-small" $\varsigma_{\min}(i)$ and $r(i+1)$ guarantee a "not-too-small" $\varsigma_{\min}(i+1)$. In other words, a stable system at time i and a "good" data point $\mathbf{u}(i+1)$ are a sufficient condition to have a stable system at time $i+1$.

REFERENCES

T. Adali, X. Liu, and M. K. Sonmez. Conditional distribution learning with neural networks and its application to channel equalization. *IEEE Transactions on Signal Processing*, 45:1051–1064, Apr. 1997.

M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.

D. L. Alspach and H. W. Sorenson. Nonlinear Baysian estimation using Gaussian sum approximations. *IEEE Transactions on Automatic Control*, 17(4):439–448, 1972.

S. An, W. Liu, and S. Venkatesh. Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression. *Pattern Recognition*, 40:2154–2162, 2007.

I. Arasaratnam and S. Haykin. Cubature Kalman filters. *IEEE Transactions on Automatic Control*, 54(6):1254–1269, 2009.

A. Argyriou, C. A. Micchelli, and M. Pontil. Learning convex combinations of continuously parameterized basic kernels. In *Proceedings of the 18th Conference on Learning Theory*, pages 338–352, 2005.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

J. F. Barrett. The use of functionals in the analysis of non-linear physical systems. *International Journal of Electronics*, 15:567–615, 1963.

S. Benedetto and E. Biglieri. Nonlinear equalization of digital satellite channels. *IEEE Journal on Selected Areas in Communications*, 1:57–62, Jan. 1983.

A. Beuter, L. Glass, M. Mackey, and M. S. Titcombe. *Nonlinear Dynamics in Physiology and Medicine*. New York: Springer, 2003.

S. Billings and S. Fakhouri. Identification of systems containing linear dynamics and static nonlinear elements. *Automatica*, 18:15–26, 1982.

S. A. Billings and C. F. Fung. Recurrent radial basis function networks for adaptive noise cancellation. *Neural Networks*, 8:273–290, June 1995.

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, 2005.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

L. Bottou. Large-scale machine learning and stochastic algorithms, 2008. Tutorial on NIPS 2008.

P. Brazdil. Learning in multi-agent environments. In *Proceedings of the European Working Session on Learning*, pages 598–605, 1991.

D. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

C. Campbell, N. Cristianini, and A. Smola. Query learning with large margin classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, pages 111–118, 2000.

G. A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics and Image Processing*, 37:54–115, 1987.

G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69:143–167, 2007.

G. C. Cawley and N. L. C. Talbot. Efficient leave-one-out cross-validation of kernel Fisher discriminant classifiers. *Patten Recognition*, 36:2585–2592, 2003.

I. Cha and S. Kassam. Channel equalization using adaptive complex radial basis function networks. *IEEE Journal on Selected Areas in Communications*, 13:122–131, Jan. 1995.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46:131–159, 2002.

S. Chen, B. Mulgrew, and P. M. Grant. A clustering technique for digital communications channel equalization using radial basis function networks. *IEEE Transactions on Neural Networks*, 4:570–590, July 1993a.

S. Chen, B. Mulgrew, and S. McLaughlin. Adaptive Bayesian equalizer with decision feedback. *IEEE Transactions on Signal Processing*, 41:2918–2927, Sept. 1993b.

Y.-H. Cheng and C.-S. Lin. A learning algorithm for radial basis function networks: with the capability of adding and pruning neurons. In *Proceedings of IEEE International Conference on Neural Networks 1994*, pages 797–801, 1994.

V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods*. New York: Wiley-Interscience, 1998.

J. A. Cherry. *Distortion Analysis of Weakly Nonlinear Filters Using Volterra Series*. Master's Thesis, Carleton University, 1994.

M. Coker and D. Simkins. A nonlinear adaptive noise canceller. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing 1980*, volume 5, pages 470–473, 1980.

T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. New York: Wiley-Interscience, 1991.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge, MA: Cambridge University Press, 2000.

R. S. Crowder. Predicting the Mackey-Glass time series with cascade-correlation learning. In *Proceedings of 1990 Connectionist Models Summer School*, 1990.

L. Csato and M. Opper. Sparse online Gaussian processes. *Neural Computation*, 14:641–668, 2002.

O. Dekel, S. Shalev-Shwartz, and Y. Singer. The Forgetron: A kernel-based perceptron on a fixed budget. In *Advances in Neural Information Processing Systems 18*, pages 1342–1372, Cambridge, MA: MIT Press, 2006.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

C. Dima and M. Hebert. Active learning for outdoor obstacle detection. In *Proceedings of Science and Systems I*, Aug. 2005.

A. Doucet, C. Andrieu, and S. Godsill. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000a.

A. Doucet, J. F. G. de Freitas, and N. J. Gordon. *Sequential Monte Carlo Methods in Practice*, 3rd edition. New York: Springer-Verlag, 2000b.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. New York: Wiley, Interscience, 2000.

M. A. El-Gamal. The role of priors in active Bayesian learning in the sequential statistical decision framework. In J. W. T. Grandy and L. H. Schick, editors, *Maximum Entropy and Bayesian Methods*, pages 33–38. Reading, MA: Kluwer, 1991.

Y. Engel, S. Mannor, and R. Meir. The kernel recursive least-squares algorithm. *IEEE Transactions on Signal Processing*, 52(8):2275–2285, 2004.

H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. New York: Springer, 2000.

M. J. Er, Z. Li, H. Cai, and Q. Chen. Adaptive noise cancellation using enhanced dynamic fuzzy neural networks. *IEEE Transactions on Fuzzy Systems*, 13:331–342, June 2005.

D. Erdogmus, D. Rende, J. C. Principe, and T. F. Wong. Nonlinear channel equalization using multilayer perceptrons with information-theoretic criterion. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing 2001*, pages 443–451, 2001.

T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. In *Advances in Computational Mathematics*, pages 1–50. Cambridge, MA: MIT Press, 2000.

S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pages 524–532. New York: Morgan Kaufmann, 1990.

J. D. Farmer. Chaotic attractors of an infinite-dimensional dynamical system. *Physica D*, 4(3), 1982.

J. D. Farmer and J. J. Sidorowich. Predicting chaotic time series. *Physical Review Letter*, 59(8):845–848, 1987.

V. V. Fedorov. *Theory of Optimal Experiment*. New York: Academic Press, 1972.

S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:242–264, 2001.

M. O. Franz and B. Schölkopf. A unifying view of Wiener and Volterra theory and polynomial kernel regression. *Neural Computation*, 18:3097–3118, 2006.

T.-T. Frieb and R. F. Harrison. A kernel-based ADALINE. In *Proceedings European Symposium on Artificial Neural Networks 1999*, pages 245–250, Apr. 1999.

K. Fukumizu. Active learning in multilayer perceptrons. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 295–301. Cambridge, MA: MIT Press, 1996.

D. Gabor. Holographic model of temporal recall. *Nature*, 217:584–585, 1968.

D. Gabor, W. P. L. Wilby, and R. Woodcock. A universal non-linear filter, predictor, and simulator which optimizes itself by a learning process. In *Proceedings of the Institution of Electrical Engineers*, volume 108, pages 422–435, 1960.

S. L. Gay and S. Tavathia. The fast affine projection algorithm. In *Proceedings of the International Conference on Accoustics, Speech and Signal Processing 1995*, volume 5, pages 3023–3026, 1995.

A. Gersho. Adaptive equalization of highly dispersive channels for data transmission. *Bell System Technical Journal*, 48:55–70, 1969.

C. Giraud-Carrier. A note on the utility of incremental learning. *AI Communications*, 13(4):215–233, 2000.

F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7:219–269, 1995.

T. Glasmachers. Second-order SMO improves SVM online and active learning. *Neural Computation*, 20:374–382, 2008.

L. Glass and M. Mackey. *From Clocks to Chaos: The Rhythms of Life*. Princeton, NJ: Princeton University Press, 1988.

G. Golub and C. Loan. *Matrix Computations*. Washington, DC: John Hopkins University Press, 1996.

G. Goodwin and K. Sin. *Adaptive Filtering: Prediction and Control*. Upper Saddle River, NJ: Prentice Hall, 1984.

N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F: Radar and Signal Processing*, 140(5):107–113, 1993.

S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11:23–63, 1987.

P. D. Grünwald. *The Minimum Description Length Principle*. Cambridge, MA: MIT Press, 2007.

K. Hagiwara and K. Kuno. Regularization learning and early stopping in linear networks. In *Proceedings of International Joint Conference on Neural Networks 2000*, volume 4, pages 511–516, 2000.

W. Härdle. *Applied Nonparametric Regression*. Cambridge, UK: Cambridge University Press, 1992.

B. Hassibi and G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Neural Information Processing Systems*, pages 164–171, 1992.

S. Haykin. *Neural Networks: A Comprehensive Foundation*, 1st edition. Upper Saddle River, NJ: Prentice Hall, 1994.

S. Haykin. *Adaptive Filter Theory*, 3rd edition. Upper Saddle River, NJ: Prentice Hall, 1996.

S. Haykin. *Neural Networks: A Comprehensive Foundation*, 2nd edition. Upper Saddle River, NJ: Prentice Hall, 1998.

S. Haykin. *Adaptive Filter Theory*, 4th edition. Upper Saddle River, NJ: Prentice Hall, 2002.

S. Haykin. *Neural Networks and Learning Machines*, 3rd edition. Upper Saddle River, NJ: Prentice Hall, 2009.

S. Haykin, A. H. Sayed, J. R. Zeidler, P. Yee, and P. C. Wei. Adaptive tracking of linear time-variant systems by extended RLS algorithms. *IEEE Transactions on Signal Processing*, 45:1118–1128, 1997.

R. Herbrich. *Learning Kernel Classifiers: Theory and Algorithms*. Cambridge, MA: MIT Press, 2002.

E. Herrmann. Local bandwidth choice in kernel regression estimation. *Journal of Computational and Graphical Statistics*, 6(1):35–54, 1997.

G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing*, volume 1, pages 282–317. Cambridge, MA: MIT Press, 1986.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

G. Huang, P. Saratchandran, and N. Sundararajan. A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. *IEEE Transactions on Neural Networks*, 16:57–67, 2005.

K. Ito and K. Xiong. Gaussian filters for nonlinear filtering problems. *IEEE Transactions on Automatic Control*, 45(5):910–927, 2000.

L. Itti and P. Baldi. Bayesian surprise attracts human attention. In *Advances in Neural Information Processing Systems, Vol. 19*, pages 1–8. Cambridge, MA: MIT Press, 2006.

A. G. Journel and C. J. Huijbregts. *Mining Geostatistics*. New York: Academic Press, 1978.

S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *International Symposium of the Aerospace/Defense Sensing, Simul. and Controls*, 1997.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME—Journal of Basic Engineering*, 82:35–45, 1960.

R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Transactions of the ASME—Journal of Basic Engineering*, 83:95–107, 1961.

N. B. Karayiannis and G. W. Mi. Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques. *IEEE Transactions on Neural Networks*, 8:1492–1506, 1997.

G. Kechriotis, E. Zarvas, and E. S. Manolakos. Using recurrent neural networks for adaptive communication channel equalization. *IEEE Transactions on Neural Networks*, 5:267–278, March 1994.

J. L. Kelly and R. F. Logan. Self-adaptive echo canceller. U.S. Patent 3,500,000, 1970.

J. Kivinen, A. Smola, and R. C. Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176, Aug. 2004.

A. N. Kolmogorov. Interpolation und extrapolation von stationären zufäligen folgen. *Izv. Akad. Nauk SSSR*, 5:3–14, 1941.

K. J. Lang and G. E. Hinton. The development of the time-delay neural network architecture for speech recognition. *Tech. report CMU-CS-88-152*, Carnegie-Mellon University, Pittsburgh, PA, 1988.

S. L. Lauritzen. Time series analysis in 1880: A discussion of contributions made by T. N. Thiele. *International Statistical Review*, 49:319–333, 1981.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2*, pages 598–605, New York: Morgan Kaufmann, 1990.

T. C. Lee, A. M. Peterson, and J. C. Tsai. Amulti-layer feed-forward neural network with dynamically adjustable structures. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 367–369, 1990.

T. Li and J. Jiang. Adaptive Volterra filters for active control of nonlinear noise processes. *IEEE Transactions on Signal Processing*, 49:1667–1676, Aug. 2001.

W. A. Light. Some aspects of radial basis functions approximation. In S. P. Singh, editor, *Approximation Theory, Spline Functions and Applications*, pages 163–190. Reading, MA: Kluwer, 1992.

M. Lindenbaum. Selective sampling for nearest neighbor classifiers. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, pages 366–371, 1999.

D. V. Lindley. On the measure of information provided by an experiment. *The Annals of Mathematical Statistics*, 27(4):986–1005, 1956.

W. Liu and J. C. Príncipe. Extended recursive least squares algorithm in RKHS. In *1st IAPR Workshop on Cognitive Information Processing*, 2008a.

W. Liu and J. C. Príncipe. The kernel affine projection algorithms. *EURASIP Journal on Advances in Signal Processing*, 2008, 2008b. URL http://www.hindawi.com/GetArticle.aspx?doi=10.1155/2008/784292.

W. Liu, P. Pokharel, and J. C. Príncipe. The kernel least mean square algorithm. *IEEE Transactions on Signal Processing*, 56(2):543–554, 2008.

W. Liu, I. Park, Y. Wang, and J. C. Príncipe. Extended kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing*, 57(10), October 2009.

E. N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20:130–141, 1963.

R. W. Lucky. Automatic equalization for digital communication. *Bell System Technical Journal*, 44:547–588, 1965.

D. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, 1992a.

D. MacKay. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992b.

M. C. Mackey and L. Glass. Pathological physiological conditions resulting from instabilities in physiological control system. *Science*, 197(4300):287–289, 1977.

V. Z. Marmarelis. Identification of nonlinear biological systems using Laguerre expansions of kernels. *Annals of Biomedical Engineering*, 21:573–589, 1993.

T. Martinetz and K. Schulten. A "Neural-Gas" network learns topologies. *Artificial Neural Networks*, I:397–402, 1991.

T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. "Neural-Gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, 1993.

G. Matheron. The intrinsic random functions and their applications. *Advances in Applied Probability*, 5:439–468, 1973.

J. M. McCool, B. Widrow, J. R. Zeidler, R. H. Hearn, and D. M. Chabries. Adaptive line enhancer. U.S. Patent 4,238,746, 1980.

C. Micchelli and M. Pontil. Learning the kernel function via regularization. *Journal of Machine Learning Research*, 6:1099–1125, 2005.

C. A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

S. Mika, G. Ratsch, J. Weston, B. Schölkopf, and K. R. Müller. Fisher discriminant analysis with kernels. In *Proceedings of IEEE Neural Networks for Signal Processing Workshop 1999*, pages 8–10, 1999.

M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.

J. E. Moody and T. S. Rögnvaldsson. Smoothing regularizers for projective basis function networks. In *Advances in Neural Information Processing Systems 9*, pages 585–591. Cambridge, MA: MIT Press, 1997.

S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear prediction of chaotic time series using support vector machines. In J. Príncipe, L. Giles, N. Morgan, and E. Wilson, editors, *IEEE Workshop on Neural Networks for Signal Processing VII*, pages 511–519, 1997.

K. R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *Proceedings of ICANN 1997*, pages 999–1004, 1997.

A. O'Hagan. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society B*, 40:1C42, 1978.

K. Ozeki and T. Umeda. An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties. *Electronics and Communications in Japan*, 67-A(5), 1984.

G. Palm. Evidence, information, and surprise. *Biological Cybernetics*, 42(1):57–68, 1981.

E. Parzen. Statistical methods on time series by Hilbert space methods. *Technical Report* 23, Stanford University, CA, 1959.

R. L. Plackett. Some theorems in least squares. *Biometrika*, 37:149–157, 1950.

J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of IEEE*, 78(9):1481–1497, 1990.

T. Poggio and S. Smale. The mathematics of learning: Dealing with data. *Transactions of the American Mathematical Society*, 50:537–544, Nov. 2003.

P. Pokharel, W. Liu, and J. C. Príncipe. Kernel LMS. In *Proceedings of the International Conference on Accoustics, Speech and Signal Processing 2007*, volume 3, pages 1421–1424, 2007.

P. Pokharel, W. Liu, and J. C. Príncipe. Kernel least mean square algorithm with constrained growth. *Signal Processing*, 89:257–265, 2009.

M. J. D. Powell. Radial basis functions for multivariable interpolation: A review. In *Clarendon Press Institute Of Mathematics And Its Applications Conference Series*, pages 143–167, 1985.

J. C. Príncipe, B. de Vries, J. M. Kuo, and P. G. de Oliveira. Modeling applications with the focused gamma net. In *Advances in Neural Information Processing Systems*, volume 4, pages 143–150, 1992.

J. Proakis. *Digital Communications*. New York: McGraw-Hill, 2000.

J. Proakis and J. Miller. An adaptive receiver for digital signaling through channels with intersymbol interference. *IEEE Transactions on Information Theory*, 15:484–497, 1969.

J. Quinonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

J. Racine. An efficient cross-validation algorithm for window width selection for nonparametric kernel regression. *Communications in Statistics: Simulation and Computation*, 22:1107–1114, 1993.

L. Ralaivola and F. d'Alche Buc. Time series filtering, smoothing and learning using the kernel Kalman filter. In *Proceedings 2005 IEEE International Joint Conference on Neural Networks*, pages 1449–1454, 2005.

N. Ranasinghe and W.-M. Shen. Surprise-based learning for developmental robotics. In *2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems*, pages 65–70, 2008.

C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006.

S. Raudys and T. Cibas. Regularization by early stopping in single layer perceptron training. In *Proceedings of the 1996 International Conference on Artificial Neural Networks*, volume 1112, pages 77–82, 1996.

C. Richard, J. C. M. Bermudez, and P. Honeine. Online prediction of time series data with kernels. *IEEE Transactions on Signal Processing*, 57(3):1058–1066, 2009.

R. M. Rifkin. *Everything Old Is New Again: A Fresh Look at Historical Approaches in Machine Learning*. PhD Dissertation, MIT, Cambridge, MA, 2002.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

G. Rombouts and M. Moonen. A fast exact frequency domain implementation of the exponentially windowed affine projection algorithm. In *Proceedings. IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, pages 342–346, 2000.

S. Roweis and Z. Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11(2), 1999.

J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, 1989.

N. P. Sands and J. M. Cioffi. Nonlinear channel models for digital magnetic recording. *IEEE Transactions on Magnetics*, 29:3996–3998, Nov. 1993.

S. G. Sankaran and A. A. L. Beex. Convergence behavior of affine projection algorithms. *IEEE Transaction on Signal Processing*, 48(4):1086–1096, 2000.

A. Sayed. *Fundamentals of Adaptive Filtering*. New York: Wiley, 2003.

A. Sayed and T. Kailath. A state-space approach to adaptive RLS filtering. *IEEE Signal Processing Magazine*, 11:18–60, 1994.

T. S. Schei. A finite difference method for linearizing in nonlinear estimation algorithms. *Automatica*, 33(11):2051–2058, 1997.

M. Schetzen. *The Volterra and Wiener Theories of Nonlinear Systems*. Malabar, FL: Krieger Publishing, 2006.

B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. Cambridge, MA: MIT Press, 2002.

B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

B. Schölkopf, R. Herbrich, A. Smola, and R. Williamson. A generalized representer theorem. In *Proceedings of the Annual Conference on Computational Learning Theory*, pages 416–426, 2001.

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

D. J. Sebald and J. A. Bucklew. Support vector machine techniques for nonlinear equalization. *IEEE Journal on Selected Areas in Communications*, 48:3217–3226, Nov. 2000.

M. Seeger and C. Williams. Fast forward selection to speed up sparse Gaussian process regression. In *Workshop on AI and Statistics 9*, 2003.

C. E. Shannon. A mathematical theory of communication. In *Bell System Technical Journal*, 379–423, July 1948.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge, UK: Cambridge University Press, 2004.

J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. *Technical report*, Carnegie Mellon University, Pittsburgh, PA, 1994. http://www.cs.cmu.edu/~jrs/jrspapers.html.

M. Simandl, J. Kralovec, and T. Söderstörm. Advanced point-mass method for nonlinear state estimation. *Automatica*, 42(7):1133–1145, 2006.

H. Simon. Why should machines learn? In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*. Wellsboro, PA: Tioga Publishing Company, 1983.

K. Slavakis and S. Theodoridis. Sliding window generalized kernel affine projection algorithm using projection mappings. *EURASIP Journal on Advances in Signal Processing*, 2008. http://www.hindawi.com/GetArticle.aspx?doi=10.1155/2008/735351.

A. J. Smola and P. L. Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems* 13, pages 619–625, 2001.

M. M. Sondhi. An adaptive echo canceller. *Bell System Technical Journal*, 46:497–511, 1967.

M. M. Sondhi. Closed loop adaptive echo canceller using generalized filter networks. U.S. Patent 3,499,999, 1970.

H. W. Sorenson. *Kalman Filtering: Theory and Application*. New York: IEEE Press, 1985.

J. Stapleton and S. Bass. Adaptive noise cancellation for a class of nonlinear, dynamic reference channels. *IEEE Transactions on Circuits and Systems*, 32:143–150, Feb. 1985.

I. Steinwart. On the influence of the kernel on the consistency of support vector machines. *Journal of Machine Learning Research*, 2:67–93, 2001.

S. M. Stigler. Gergonne's 1815 paper on the design and analysis of polynomial regression experiments. *Historia Mathematical 1*, 4:431–439, 1974.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

J. A. K. Suykens, L. Lukas, and J. Vandewalle. Sparse approximation using least squares support vector machines. In *IEEE International Symposium on Circuits and Systems 2000*, pages 757–760, 2000.

F. Takens. Detecting strange attractors in turbulence. *Dynamical Systems and Turbulence*, pages 366–381, 1981.

M. Tan. Adaptation-based cooperative learning multiagent systems. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

M. Tanaka, S. Makino, and J. Kojima. A block exact fast affine projection algorithm. *IEEE Transactions on Speech and Audio Processing*, 7:79–86, 1999.

P. Tans. Trends in atmospheric carbon dioxide—Mauna Loa. NOAA/ESRL, 2008. www.esrl.noaa.gov/gmd/ccgg/trends.

S. Theodoridis, C. M. S. See, and C. F. N. Cowan. Nonlinear channel equalization using clustering techniques. In *Proceedings of IEEE International Conference on Communications 1992*, volume 3, pages 1277–1279, 1992.

P. D. Thompson. Optimum smoothing of two-dimensional fields. *Tellus*, 8:384–393, 1956.

A. Tikhonov and V. Arsenin. *Solution of Ill-posed Problems*. Washington, DC: Winston & Sons, 1977.

M. E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In P. Langley, editor, *Proceedings of 17th International Conference on Machine Learning*, pages 999–1006, 2000.

W. Tucker. A Rigorous ODE Solver and Smale's 14th Problem. *Foundations of Computational Mathematics*, 2:53–117, 2002.

S. Van Vaerenbergh, J. Via, and I. Santamaría. A sliding-window kernel RLS algorithm and its application to nonlinear channel identification. In *Proceedings of the International Conference on Accoustics, Speech and Signal Processing 2006*, volume 5, pages 789–792, May 2006.

V. Vapnik. *The Nature of Statistical Learning Theory*. New York: Springer Verlag, 1995.

V. Vapnik. *Statistical Learning Theory*. New York: Wiley, 1998.

V. Volterra. Sopra le funzioni che dipendono de altre funzioni. *Rend. R. Academia dei Lincei 2 Sem.*, 59:97–105, 1887.

G. Wahba. *Spline Models for Observational Data*. Philadelphia, PA: SIAM, 1990.

E. A. Wan. Temporal backpropogation for FIR neural networks. In *IEEE International Joint Conference on Neural Networks*, volume I, pages 575–580, 1990.

A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing Systems 3*, pages 875–882. San Francisco, CA: Morgan Kaufmann, 1990.

P. Whittle. *Prediction and Regulation by Linear Least-Square Methods*. Oxford, UK: English Universities Press, 1963.

B. Widrow and M. E. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record*, 4:96–104, 1960.

B. Widrow and S. D. Stearns. *Adaptive Signal Processing*. Upper Saddle River, NJ: Prentice-Hall, 1985.

B. Widrow, J. Glover, J. R., J. McCool, J. Kaunitz, C. Williams, R. Hearn, J. Zeidler, J. Eugene Dong, and R. Goodlin. Adaptive noise cancelling: Principles and applications. *Proceedings of the IEEE*, 63:1692–1716, 1975.

N. Wiener. *Extrapolation, Interpolation and Smoothing of Stationary Time Series*. Cambridge, MA: MIT Press, 1949.

N. Wiener. *Nonlinear Problems in Random Theory*. New York: Wiley, 1958.

C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, pages 682–688. Cambridge, UK: MIT Press, 2001.

C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 514–520. Cambridge, UK: MIT Press, 1996.

G. Williams. *Linear Algebra with Applications*. Sudbury, MA: Jones and Bartlett Publishers, 2007.

R. L. Winkler. *Introduction to Bayesian Inference and Decision*, 2nd edition. Gainesville, FL: Probabilistic Publishing, 2003.

Y. Yao, L. Rosasco, and A. Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

INDEX

Kernel Adaptive Filtering, By Weifeng Liu, José C. Príncipe, and Simon Haykin

Adaptive and Learning Systems for Signal Processing, Communication, and Control

Editor: Simon Haykin

Adali and Haykin / ADAPTIVE SIGNAL PROCESSING: Next Generation Solutions

Beckerman / ADAPTIVE COOPERATIVE SYSTEMS

Candy / BAYESIAN SIGNAL PROCESSING: CLASSICAL, MODERN, AND PARTICLE FILTERING METHODS

Candy / MODEL-BASED SIGNAL PROCESSING

Chen and Gu / CONTROL-ORIENTED SYSTEM IDENTIFICATION: An $\mathcal{H}_\infty$ Approach

Chen, Haykin, Eggermont, and Becker / CORRELATIVE LEARNING: A Basis for Brain and Adaptive Systems

Cherkassky and Mulier / LEARNING FROM DATA: Concepts, Theory, and Methods

Costa and Haykin / MULTIPLE-INPUT MULTIPLE-OUTPUT CHANNEL MODELS: Theory and Practice

Diamantaras and Kung / PRINCIPAL COMPONENT NEURAL NETWORKS: Theory and Applications

Farrell and Polycarpou / ADAPTIVE APPROXIMATION BASED CONTROL: Unifying Neural, Fuzzy and Traditional Adaptive Approximation Approaches

Gini and Rangaswamy / KNOWLEDGE-BASED RADAR DETECTION: Tracking and Classification

Hänsler and Schmidt / ACOUSTIC ECHO AND NOISE CONTROL: A Practical Approach

Haykin / UNSUPERVISED ADAPTIVE FILTERING: Blind Source Separation

Haykin / UNSUPERVISED ADAPTIVE FILTERING: Blind Deconvolution

Haykin and Puthussarypady / CHAOTIC DYNAMICS OF SEA CLUTTER

Haykin and Widrow / LEAST-MEAN-SQUARE ADAPTIVE FILTERS

Hrycej / NEUROCONTROL: Towards an Industrial Control Methodology

Hyvärinen, Karhunen, and Oja / INDEPENDENT COMPONENT ANALYSIS

Kristić, Kanellakopoulos, and Kokotović / NONLINEAR AND ADAPTIVE CONTROL DESIGN

Liu, Príncipe, and Haykin / KERNAL ADAPTIVE FILTERING

Mann / INTELLIGENT IMAGE PROCESSING

Nikias and Shao / SIGNAL PROCESSING WITH ALPHA-STABLE DISTRIBUTIONS AND APPLICATIONS

Passino and Burgess / STABILITY ANALYSIS OF DISCRETE EVENT SYSTEMS

Sánchez-Peña and Sznaier / ROBUST SYSTEMS THEORY AND APPLICATIONS

Sandberg, Lo, Fancourt, Principe, Katagiri, and Haykin / NONLINEAR DYNAMICAL SYSTEMS: Feedforward Neural Network Perspectives

Sellathurai and Haykin / SPACE-TIME LAYERED INFORMATION PROCESSING FOR WIRELESS COMMUNICATIONS

Spooner, Maggiore, Ordóñez, and Passino / STABLE ADAPTIVE CONTROL AND ESTIMATION FOR NONLINEAR SYSTEMS: Neural and Fuzzy Approximator Techniques

Tao / ADAPTIVE CONTROL DESIGN AND ANALYSIS

Tao and Kokotović / ADAPTIVE CONTROL OF SYSTEMS WITH ACTUATOR AND SENSOR NONLINEARITIES

Tsoukalas and Uhrig / FUZZY AND NEURAL APPROACHES IN ENGINEERING

Van Hulle / FAITHFUL REPRESENTATIONS AND TOPOGRAPHIC MAPS: From Distortion- to Information-Based Self-Organization

Vapnik / STATISTICAL LEARNING THEORY

Werbos / THE ROOTS OF BACKPROPAGATION: From Ordered Derivatives to Neural Networks and Political Forecasting

Yee and Haykin / REGULARIZED RADIAL BIAS FUNCTION NETWORKS: Theory and Applications