

Matthew F. Dixon
Igor Halperin
Paul Bilokon

Machine Learning in Finance

From Theory to Practice



Machine Learning in Finance

Matthew F. Dixon • Igor Halperin • Paul Bilokon

Machine Learning in Finance

From Theory to Practice



Springer

Matthew F. Dixon
Department of Applied Mathematics
Illinois Institute of Technology
Chicago, IL, USA

Igor Halperin
Tandon School of Engineering
New York University
Brooklyn, NY, USA

Paul Bilokon
Department of Mathematics
Imperial College London
London, UK

Additional material to this book can be downloaded from http://mypages.iit.edu/~mdixon7/book/ML_Finance_Codes-Book.zip

ISBN 978-3-030-41067-4 ISBN 978-3-030-41068-1 (eBook)
<https://doi.org/10.1007/978-3-030-41068-1>

© Springer Nature Switzerland AG 2020

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth.

—Arthur Conan Doyle

Introduction

Machine learning in finance sits at the intersection of a number of emergent and established disciplines including pattern recognition, financial econometrics, statistical computing, probabilistic programming, and dynamic programming. With the trend towards increasing computational resources and larger datasets, machine learning has grown into a central computational engineering field, with an emphasis placed on plug-and-play algorithms made available through open-source machine learning toolkits. Algorithm focused areas of finance, such as algorithmic trading have been the primary adopters of this technology. But outside of engineering-based research groups and business activities, much of the field remains a mystery.

A key barrier to understanding machine learning for non-engineering students and practitioners is the absence of the well-established theories and concepts that financial time series analysis equips us with. These serve as the basis for the development of financial modeling intuition and scientific reasoning. Moreover, machine learning is heavily entrenched in engineering ontology, which makes developments in the field somewhat intellectually inaccessible for students, academics, and finance practitioners from the quantitative disciplines such as mathematics, statistics, physics, and economics. Consequently, there is a great deal of misconception and limited understanding of the capacity of this field. While machine learning techniques are often effective, they remain poorly understood and are often mathematically indefensible. How do we place key concepts in the field of machine learning in the context of more foundational theory in time series analysis, econometrics, and mathematical statistics? Under which simplifying conditions are advanced machine learning techniques such as deep neural networks mathematically equivalent to well-known statistical models such as linear regression? How should we reason about the perceived benefits of using advanced machine learning methods over more traditional econometrics methods, for different financial applications? What theory supports the application of machine learning to problems in financial modeling? How does reinforcement learning provide a model-free approach to the Black–Scholes–Merton model for derivative pricing? How does Q-learning generalize discrete-time stochastic control problems in finance?

This book is written for advanced graduate students and academics in financial econometrics, management science, and applied statistics, in addition to quants and data scientists in the field of quantitative finance. We present machine learning as a non-linear extension of various topics in quantitative economics such as financial econometrics and dynamic programming, with an emphasis on novel algorithmic representations of data, regularization, and techniques for controlling the bias-variance tradeoff leading to improved out-of-sample forecasting. The book is presented in three parts, each part covering theory and applications. The first part presents supervised learning for cross-sectional data from both a Bayesian and frequentist perspective. The more advanced material places a firm emphasis on neural networks, including deep learning, as well as Gaussian processes, with examples in investment management and derivatives. The second part covers supervised learning for time series data, arguably the most common data type used in finance with examples in trading, stochastic volatility, and fixed income modeling. Finally, the third part covers reinforcement learning and its applications in trading, investment, and wealth management. We provide Python code examples to support the readers' understanding of the methodologies and applications. As a bridge to research in this emergent field, we present the frontiers of machine learning in finance from a researcher's perspective, highlighting how many well-known concepts in statistical physics are likely to emerge as research topics for machine learning in finance.

Prerequisites

This book is targeted at graduate students in data science, mathematical finance, financial engineering, and operations research seeking a career in quantitative finance, data science, analytics, and fintech. Students are expected to have completed upper section undergraduate courses in linear algebra, multivariate calculus, advanced probability theory and stochastic processes, statistics for time series (econometrics), and gained some basic introduction to numerical optimization and computational mathematics. Students shall find the later chapters of this book, on reinforcement learning, more accessible with some background in investment science. Students should also have prior experience with Python programming and, ideally, taken a course in computational finance and introductory machine learning. The material in this book is more mathematical and less engineering focused than most courses on machine learning, and for this reason we recommend reviewing the recent book, *Linear Algebra and Learning from Data* by Gilbert Strang as background reading.

Advantages of the Book

Readers will find this book useful as a bridge from well-established foundational topics in financial econometrics to applications of machine learning in finance. Statistical machine learning is presented as a non-parametric extension of financial econometrics and quantitative finance, with an emphasis on novel algorithmic representations of data, regularization, and model averaging to improve out-of-sample forecasting. The key distinguishing feature from classical financial econometrics and dynamic programming is the absence of an assumption on the data generation process. This has important implications for modeling and performance assessment which are emphasized with examples throughout the book. Some of the main contributions of the book are as follows:

- The textbook market is saturated with excellent books on machine learning. However, few present the topic from the prospective of financial econometrics and cast fundamental concepts in machine learning into canonical modeling and decision frameworks already well established in finance such as financial time series analysis, investment science, and financial risk management. Only through the integration of these disciplines can we develop an intuition into how machine learning theory informs the practice of financial modeling.
- Machine learning is entrenched in engineering ontology, which makes developments in the field somewhat intellectually inaccessible for students, academics, and finance practitioners from quantitative disciplines such as mathematics, statistics, physics, and economics. Moreover, financial econometrics has not kept pace with this transformative field, and there is a need to reconcile various modeling concepts between these disciplines. This textbook is built around powerful mathematical ideas that shall serve as the basis for a graduate course for students with prior training in probability and advanced statistics, linear algebra, times series analysis, and Python programming.
- This book provides financial market motivated and compact theoretical treatment of financial modeling with machine learning for the benefit of regulators, wealth managers, federal research agencies, and professionals in other heavily regulated business functions in finance who seek a more theoretical exposition to allay concerns about the “black-box” nature of machine learning.
- Reinforcement learning is presented as a model-free framework for stochastic control problems in finance, covering portfolio optimization, derivative pricing, and wealth management applications without assuming a data generation process. We also provide a model-free approach to problems in market microstructure, such as optimal execution, with Q-learning. Furthermore, our book is the first to present on methods of inverse reinforcement learning.
- Multiple-choice questions, numerical examples, and more than 80 end-of-chapter exercises are used throughout the book to reinforce key technical concepts.

- This book provides Python codes demonstrating the application of machine learning to algorithmic trading and financial modeling in risk management and equity research. These codes make use of powerful open-source software toolkits such as Google’s TensorFlow and Pandas, a data processing environment for Python.

Overview of the Book

Chapter 1

Chapter 1 provides the industry context for machine learning in finance, discussing the critical events that have shaped the finance industry’s need for machine learning and the unique barriers to adoption. The finance industry has adopted machine learning to varying degrees of sophistication. How it has been adopted is heavily fragmented by the academic disciplines underpinning the applications. We view some key mathematical examples that demonstrate the nature of machine learning and how it is used in practice, with the focus on building intuition for more technical expositions in later chapters. In particular, we begin to address many finance practitioner’s concerns that neural networks are a “black-box” by showing how they are related to existing well-established techniques such as linear regression, logistic regression, and autoregressive time series models. Such arguments are developed further in later chapters.

Chapter 2

Chapter 2 introduces probabilistic modeling and reviews foundational concepts in Bayesian econometrics such as Bayesian inference, model selection, online learning, and Bayesian model averaging. We develop more versatile representations of complex data with probabilistic graphical models such as mixture models.

Chapter 3

Chapter 3 introduces Bayesian regression and shows how it extends many of the concepts in the previous chapter. We develop kernel-based machine learning methods—specifically Gaussian process regression, an important class of Bayesian machine learning methods—and demonstrate their application to “surrogate” models of derivative prices. This chapter also provides a natural point from which to

develop intuition for the role and functional form of regularization in a frequentist setting—the subject of subsequent chapters.

Chapter 4

Chapter 4 provides a more in-depth description of supervised learning, deep learning, and neural networks—presenting the foundational mathematical and statistical learning concepts and explaining how they relate to real-world examples in trading, risk management, and investment management. These applications present challenges for forecasting and model design and are presented as a reoccurring theme throughout the book. This chapter moves towards a more engineering style exposition of neural networks, applying concepts in the previous chapters to elucidate various model design choices.

Chapter 5

Chapter 5 presents a method for interpreting neural networks which imposes minimal restrictions on the neural network design. The chapter demonstrates techniques for interpreting a feedforward network, including how to rank the importance of the features. In particular, an example demonstrating how to apply interpretability analysis to deep learning models for factor modeling is also presented.

Chapter 6

Chapter 6 provides an overview of the most important modeling concepts in financial econometrics. Such methods form the conceptual basis and performance baseline for more advanced neural network architectures presented in the next chapter. In fact, each type of architecture is a generalization of many of the models presented here. This chapter is especially useful for students from an engineering or science background, with little exposure to econometrics and time series analysis.

Chapter 7

Chapter 7 presents a powerful class of probabilistic models for financial data. Many of these models overcome some of the severe stationarity limitations of the frequentist models in the previous chapters. The fitting procedure demonstrated is also different—the use of Kalman filtering algorithms for state-space models rather

than maximum likelihood estimation or Bayesian inference. Simple examples of hidden Markov models and particle filters in finance and various algorithms are presented.

Chapter 8

Chapter 8 presents various neural network models for financial time series analysis, providing examples of how they relate to well-known techniques in financial econometrics. Recurrent neural networks (RNNs) are presented as non-linear time series models and generalize classical linear time series models such as $AR(p)$. They provide a powerful approach for prediction in financial time series and generalize to non-stationary data. The chapter also presents convolution neural networks for filtering time series data and exploiting different scales in the data. Finally, this chapter demonstrates how autoencoders are used to compress information and generalize principal component analysis.

Chapter 9

Chapter 9 introduces Markov decision processes and the classical methods of dynamic programming, before building familiarity with the ideas of reinforcement learning and other approximate methods for solving MDPs. After describing Bellman optimality and iterative value and policy updates before moving to Q-learning, the chapter quickly advances towards a more engineering style exposition of the topic, covering key computational concepts such as greediness, batch learning, and Q-learning. Through a number of mini-case studies, the chapter provides insight into how RL is applied to optimization problems in asset management and trading. These examples are each supported with Python notebooks.

Chapter 10

Chapter 10 considers real-world applications of reinforcement learning in finance, as well as further advances the theory presented in the previous chapter. We start with one of the most common problems of quantitative finance, which is the problem of optimal portfolio trading in discrete time. Many practical problems of trading or risk management amount to different forms of dynamic portfolio optimization, with different optimization criteria, portfolio composition, and constraints. The chapter introduces a reinforcement learning approach to option pricing that generalizes the classical Black–Scholes model to a data-driven approach using Q-learning. It then presents a probabilistic extension of Q-learning called G-learning and shows how it

can be used for dynamic portfolio optimization. For certain specifications of reward functions, G-learning is semi-analytically tractable and amounts to a probabilistic version of linear quadratic regulators (LQRs). Detailed analyses of such cases are presented and we show their solutions with examples from problems of dynamic portfolio optimization and wealth management.

Chapter 11

Chapter 11 provides an overview of the most popular methods of inverse reinforcement learning (IRL) and imitation learning (IL). These methods solve the problem of optimal control in a data-driven way, similarly to reinforcement learning, however with the critical difference that now rewards are *not* observed. The problem is rather to learn the reward function from the observed behavior of an agent. As behavioral data without rewards are widely available, the problem of learning from such data is certainly very interesting. The chapter provides a moderate-level description of the most promising IRL methods, equips the reader with sufficient knowledge to understand and follow the current literature on IRL, and presents examples that use simple simulated environments to see how these methods perform when we know the “ground truth” rewards. We then present use cases for IRL in quantitative finance that include applications to trading strategy identification, sentiment-based trading, option pricing, inference of portfolio investors, and market modeling.

Chapter 12

Chapter 12 takes us forward to emerging research topics in quantitative finance and machine learning. Among many interesting emerging topics, we focus here on two broad themes. The first one deals with unification of supervised learning and reinforcement learning as two tasks of perception-action cycles of agents. We outline some recent research ideas in the literature including in particular information theory-based versions of reinforcement learning and discuss their relevance for financial applications. We explain why these ideas might have interesting practical implications for RL financial models, where feature selection could be done within the general task of optimization of a long-term objective, rather than outside of it, as is usually performed in “alpha-research.”

The second topic presented in this chapter deals with using methods of reinforcement learning to construct models of market dynamics. We also introduce some advanced physics-based approaches for computations for such RL-inspired market models.

Source Code

Many of the chapters are accompanied by Python notebooks to illustrate some of the main concepts and demonstrate application of machine learning methods. Each notebook is lightly annotated. Many of these notebooks use TensorFlow. We recommend loading these notebooks, together with any accompanying Python source files and data, in Google Colab. Please see the appendices of each chapter accompanied by notebooks, and the README .md in the subfolder of each chapter, for further instructions and details.

Scope

We recognize that the field of machine learning is developing rapidly and to keep abreast of the research in this field is a challenging pursuit. Machine learning is an umbrella term for a number of methodology classes, including supervised learning, unsupervised learning, and reinforcement learning. This book focuses on supervised learning and reinforcement learning because these are the areas with the most overlap with econometrics, predictive modeling, and optimal control in finance. Supervised machine learning can be categorized as generative and discriminative. Our focus is on discriminative learners which attempt to partition the input space, either directly, through affine transformations or through projections onto a manifold. Neural networks have been shown to provide a universal approximation to a wide class of functions. Moreover, they can be shown to reduce to other well-known statistical techniques and are adaptable to time series data.

Extending time series models, a number of chapters in this book are devoted to an introduction to reinforcement learning (RL) and inverse reinforcement learning (IRL) that deal with problems of optimal control of such time series and show how many classical financial problems such as portfolio optimization, option pricing, and wealth management can naturally be posed as problems for RL and IRL. We present simple RL methods that can be applied for these problems, as well as explain how neural networks can be used in these applications.

There are already several excellent textbooks covering other classical machine learning methods, and we instead choose to focus on how to cast machine learning into various financial modeling and decision frameworks. We emphasize that much of this material is not unique to neural networks, but comparisons of alternative supervised learning approaches, such as random forests, are beyond the scope of this book.

Multiple-Choice Questions

Multiple-choice questions are included after introducing a key concept. The correct answers to all questions are provided at the end of each chapter with selected, partial, explanations to some of the more challenging material.

Exercises

The exercises that appear at the end of every chapter form an important component of the book. Each exercise has been chosen to reinforce concepts explained in the text, to stimulate the application of machine learning in finance, and to gently bridge material in other chapters. It is graded according to difficulty ranging from (*), which denotes a simple exercise which might take a few minutes to complete, through to (**), which denotes a significantly more complex exercise. Unless specified otherwise, all equations referenced in each exercise correspond to those in the corresponding chapter.

Instructor Materials

The book is supplemented by a separate Instructor's Manual which provides worked solutions to the end of chapter questions. Full explanations for the solutions to the multiple-choice questions are also provided. The manual provides additional notes and example code solutions for some of the programming exercises in the later chapters.

Acknowledgements

This book is dedicated to the late Mark Davis (Imperial College) who was an inspiration in the field of mathematical finance and engineering, and formative in our careers. Peter Carr, Chair of the Department of Financial Engineering at NYU Tandon, has been instrumental in supporting the growth of the field of machine learning in finance. Through providing speaker engagements and machine learning instructorship positions in the MS in Algorithmic Finance Program, the authors have been able to write research papers and identify the key areas required by a text book. Miquel Alonso (AIFI), Agostino Capponi (Columbia), Rama Cont (Oxford), Kay Giesecke (Stanford), Ali Hirsa (Columbia), Sebastian Jaimungal (University of Toronto), Gary Kazantsev (Bloomberg), Morton Lane (UIUC), Jörg Osterrieder (ZHAW) have established various academic and joint academic-industry workshops

and community meetings to proliferate the field and serve as input for this book. At the same time, there has been growing support for the development of a book in London, where several SIAM/LMS workshops and practitioner special interest groups, such as the Thalesians, have identified a number of compelling financial applications. The material has grown from courses and invited lectures at NYU, UIUC, Illinois Tech, Imperial College and the 2019 Bootcamp on Machine Learning in Finance at the Fields Institute, Toronto.

Along the way, we have been fortunate to receive the support of Tomasz Bielecki (Illinois Tech), Igor Cialenco (Illinois Tech), Ali Hirsa (Columbia University), and Brian Peterson (DV Trading). Special thanks to research collaborators and colleagues Kay Giesecke (Stanford University), Diego Klabjan (NWU), Nick Polson (Chicago Booth), and Harvey Stein (Bloomberg), all of whom have shaped our understanding of the emerging field of machine learning in finance and the many practical challenges. We are indebted to Sri Krishnamurthy (QuantUniversity), Saeed Amen (Cuemacro), Tyler Ward (Google), and Nicole Königstein for their valuable input on this book. We acknowledge the support of a number of Illinois Tech graduate students who have contributed to the source code examples and exercises: Xiwen Jing, Bo Wang, and Siliang Xong. Special thanks to Swaminathan Sethuraman for his support of the code development, to Volod Chernat and George Gvishiani who provided support and code development for the course taught at NYU and Coursera. Finally, we would like to thank the students and especially the organisers of the MSc Finance and Mathematics course at Imperial College, where many of the ideas presented in this book have been tested: Damiano Brigo, Antoine (Jack) Jacquier, Mikko Pakkanen, and Rula Murtada. We would also like to thank Blanka Horvath for many useful suggestions.

Chicago, IL, USA
Brooklyn, NY, USA
London, UK
December 2019

Matthew F. Dixon
Igor Halperin
Paul Bilokon

Contents

Part I Machine Learning with Cross-Sectional Data

1	Introduction	3
1	Background	3
1.1	Big Data—Big Compute in Finance	4
1.2	Fintech	6
2	Machine Learning and Prediction	8
2.1	Entropy	11
2.2	Neural Networks	14
3	Statistical Modeling vs. Machine Learning	16
3.1	Modeling Paradigms	16
3.2	Financial Econometrics and Machine Learning	18
3.3	Over-fitting	21
4	Reinforcement Learning	22
5	Examples of Supervised Machine Learning in Practice	28
5.1	Algorithmic Trading	29
5.2	High-Frequency Trade Execution	32
5.3	Mortgage Modeling	34
6	Summary	40
7	Exercises	41
	References	44
2	Probabilistic Modeling	47
1	Introduction	47
2	Bayesian vs. Frequentist Estimation	48
3	Frequentist Inference from Data	51
4	Assessing the Quality of Our Estimator: Bias and Variance	53
5	The Bias–Variance Tradeoff (Dilemma) for Estimators	55
6	Bayesian Inference from Data	56
6.1	A More Informative Prior: The Beta Distribution	60
6.2	Sequential Bayesian updates	61

6.3	Practical Implications of Choosing a Classical or Bayesian Estimation Framework	63
7	Model Selection	63
7.1	Bayesian Inference	64
7.2	Model Selection	65
7.3	Model Selection When There Are Many Models	66
7.4	Occam's Razor	69
7.5	Model Averaging	69
8	Probabilistic Graphical Models	70
8.1	Mixture Models	72
9	Summary	76
10	Exercises	76
	References	80
3	Bayesian Regression and Gaussian Processes	81
1	Introduction	81
2	Bayesian Inference with Linear Regression	82
2.1	Maximum Likelihood Estimation	86
2.2	Bayesian Prediction	88
2.3	Schur Identity	89
3	Gaussian Process Regression	91
3.1	Gaussian Processes in Finance	92
3.2	Gaussian Processes Regression and Prediction	93
3.3	Hyperparameter Tuning	94
3.4	Computational Properties	96
4	Massively Scalable Gaussian Processes	96
4.1	Structured Kernel Interpolation (SKI)	97
4.2	Kernel Approximations	97
5	Example: Pricing and Greeking with Single-GPs	98
5.1	Greeking	101
5.2	Mesh-Free GPs	101
5.3	Massively Scalable GPs	103
6	Multi-response Gaussian Processes	103
6.1	Multi-Output Gaussian Process Regression and Prediction	104
7	Summary	105
8	Exercises	106
8.1	Programming Related Questions*	107
	References	108
4	Feedforward Neural Networks	111
1	Introduction	111
2	Feedforward Architectures	112
2.1	Preliminaries	112
2.2	Geometric Interpretation of Feedforward Networks	114
2.3	Probabilistic Reasoning	117

2.4	Function Approximation with Deep Learning*	119
2.5	VC Dimension	120
2.6	When Is a Neural Network a Spline?*	124
2.7	Why Deep Networks?	127
3	Convexity and Inequality Constraints	132
3.1	Similarity of MLPs with Other Supervised Learners	138
4	Training, Validation, and Testing	140
5	Stochastic Gradient Descent (SGD)	142
5.1	Back-Propagation	143
5.2	Momentum	146
6	Bayesian Neural Networks*	149
7	Summary	153
8	Exercises	153
8.1	Programming Related Questions*	156
	References	164
5	Interpretability	167
1	Introduction	167
2	Background on Interpretability	168
2.1	Sensitivities	168
3	Explanatory Power of Neural Networks	169
3.1	Multiple Hidden Layers	170
3.2	Example: Step Test	170
4	Interaction Effects	170
4.1	Example: Friedman Data	171
5	Bounds on the Variance of the Jacobian	172
5.1	Chernoff Bounds	174
5.2	Simulated Example	174
6	Factor Modeling	177
6.1	Non-linear Factor Models	177
6.2	Fundamental Factor Modeling	178
7	Summary	183
8	Exercises	184
8.1	Programming Related Questions*	184
	References	188

Part II Sequential Learning

6	Sequence Modeling	191
1	Introduction	191
2	Autoregressive Modeling	192
2.1	Preliminaries	192
2.2	Autoregressive Processes	194
2.3	Stability	195
2.4	Stationarity	195
2.5	Partial Autocorrelations	197

2.6	Maximum Likelihood Estimation	199
2.7	Heteroscedasticity	200
2.8	Moving Average Processes	201
2.9	GARCH	202
2.10	Exponential Smoothing	204
3	Fitting Time Series Models: The Box–Jenkins Approach	205
3.1	Stationarity	205
3.2	Transformation to Ensure Stationarity	206
3.3	Identification	206
3.4	Model Diagnostics	208
4	Prediction	210
4.1	Predicting Events	210
4.2	Time Series Cross-Validation	213
5	Principal Component Analysis	213
5.1	Principal Component Projection	215
5.2	Dimensionality Reduction	216
6	Summary	217
7	Exercises	218
	Reference	220
7	Probabilistic Sequence Modeling	221
1	Introduction	221
2	Hidden Markov Modeling	222
2.1	The Viterbi Algorithm	224
2.2	State-Space Models	227
3	Particle Filtering	227
3.1	Sequential Importance Resampling (SIR)	228
3.2	Multinomial Resampling	229
3.3	Application: Stochastic Volatility Models	230
4	Point Calibration of Stochastic Filters	231
5	Bayesian Calibration of Stochastic Filters	233
6	Summary	235
7	Exercises	235
	References	237
8	Advanced Neural Networks	239
1	Introduction	239
2	Recurrent Neural Networks	240
2.1	RNN Memory: Partial Autocovariance	244
2.2	Stability	245
2.3	Stationarity	246
2.4	Generalized Recurrent Neural Networks (GRNNs)	248
3	Gated Recurrent Units	249
3.1	α -RNNs	249
3.2	Neural Network Exponential Smoothing	251
3.3	Long Short-Term Memory (LSTM)	254

4	Python Notebook Examples	255
4.1	Bitcoin Prediction.....	256
4.2	Predicting from the Limit Order Book.....	256
5	Convolutional Neural Networks	257
5.1	Weighted Moving Average Smoothers	258
5.2	2D Convolution	261
5.3	Pooling	263
5.4	Dilated Convolution	264
5.5	Python Notebooks	265
6	Autoencoders	266
6.1	Linear Autoencoders.....	267
6.2	Equivalence of Linear Autoencoders and PCA	268
6.3	Deep Autoencoders	270
7	Summary.....	271
8	Exercises	272
8.1	Programming Related Questions*	273
	References.....	275

Part III Sequential Data with Decision-Making

9	Introduction to Reinforcement Learning	279
1	Introduction	279
2	Elements of Reinforcement Learning	284
2.1	Rewards	284
2.2	Value and Policy Functions	286
2.3	Observable Versus Partially Observable Environments	286
3	Markov Decision Processes	289
3.1	Decision Policies.....	291
3.2	Value Functions and Bellman Equations	293
3.3	Optimal Policy and Bellman Optimality.....	296
4	Dynamic Programming Methods	299
4.1	Policy Evaluation	300
4.2	Policy Iteration	302
4.3	Value Iteration	303
5	Reinforcement Learning Methods	306
5.1	Monte Carlo Methods	307
5.2	Policy-Based Learning	309
5.3	Temporal Difference Learning	311
5.4	SARSA and Q-Learning.....	313
5.5	Stochastic Approximations and Batch-Mode Q-learning	316
5.6	Q-learning in a Continuous Space: Function Approximation	323
5.7	Batch-Mode Q-Learning	327
5.8	Least Squares Policy Iteration.....	331
5.9	Deep Reinforcement Learning	335

6	Summary	337
7	Exercises	337
	References	345
10	Applications of Reinforcement Learning	347
1	Introduction	347
2	The QLBS Model for Option Pricing	349
3	Discrete-Time Black–Scholes–Merton Model	352
3.1	Hedge Portfolio Evaluation	352
3.2	Optimal Hedging Strategy	354
3.3	Option Pricing in Discrete Time	356
3.4	Hedging and Pricing in the BS Limit	359
4	The QLBS Model	360
4.1	State Variables	361
4.2	Bellman Equations	362
4.3	Optimal Policy	365
4.4	DP Solution: Monte Carlo Implementation	368
4.5	RL Solution for QLBS: Fitted Q Iteration	370
4.6	Examples	373
4.7	Option Portfolios	375
4.8	Possible Extensions	379
5	G-Learning for Stock Portfolios	380
5.1	Introduction	380
5.2	Investment Portfolio	381
5.3	Terminal Condition	382
5.4	Asset Returns Model	383
5.5	Signal Dynamics and State Space	383
5.6	One-Period Rewards	384
5.7	Multi-period Portfolio Optimization	386
5.8	Stochastic Policy	386
5.9	Reference Policy	388
5.10	Bellman Optimality Equation	388
5.11	Entropy-Regularized Bellman Optimality Equation	389
5.12	G-Function: An Entropy-Regularized Q-Function	391
5.13	G-Learning and F-Learning	393
5.14	Portfolio Dynamics with Market Impact	395
5.15	Zero Friction Limit: LQR with Entropy Regularization	396
5.16	Non-zero Market Impact: Non-linear Dynamics	400
6	RL for Wealth Management	401
6.1	The Merton Consumption Problem	401
6.2	Portfolio Optimization for a Defined Contribution Retirement Plan	405
6.3	G-Learning for Retirement Plan Optimization	408
6.4	Discussion	413
7	Summary	413

8 Exercises	414
References	416
11 Inverse Reinforcement Learning and Imitation Learning	419
1 Introduction	419
2 Inverse Reinforcement Learning	423
2.1 RL Versus IRL	425
2.2 What Are the Criteria for Success in IRL?	426
2.3 Can a Truly Portable Reward Function Be Learned with IRL?	427
3 Maximum Entropy Inverse Reinforcement Learning	428
3.1 Maximum Entropy Principle	430
3.2 Maximum Causal Entropy	433
3.3 G-Learning and Soft Q-Learning	436
3.4 Maximum Entropy IRL	438
3.5 Estimating the Partition Function	442
4 Example: MaxEnt IRL for Inference of Customer Preferences	443
4.1 IRL and the Problem of Customer Choice	444
4.2 Customer Utility Function	445
4.3 Maximum Entropy IRL for Customer Utility	446
4.4 How Much Data Is Needed? IRL and Observational Noise	450
4.5 Counterfactual Simulations	452
4.6 Finite-Sample Properties of MLE Estimators	454
4.7 Discussion	455
5 Adversarial Imitation Learning and IRL	457
5.1 Imitation Learning	457
5.2 GAIL: Generative Adversarial Imitation Learning	459
5.3 GAIL as an Art of Bypassing RL in IRL	461
5.4 Practical Regularization in GAIL	464
5.5 Adversarial Training in GAIL	466
5.6 Other Adversarial Approaches*	468
5.7 f-Divergence Training*	468
5.8 Wasserstein GAN*	469
5.9 Least Squares GAN*	471
6 Beyond GAIL: AIRL, f-MAX, FAIRL, RS-GAIL, etc.*	471
6.1 AIRL: Adversarial Inverse Reinforcement Learning	472
6.2 Forward KL or Backward KL?	474
6.3 f-MAX	476
6.4 Forward KL: FAIRL	477
6.5 Risk-Sensitive GAIL (RS-GAIL)	479
6.6 Summary	481
7 Gaussian Process Inverse Reinforcement Learning	481
7.1 Bayesian IRL	482
7.2 Gaussian Process IRL	483

8	Can IRL Surpass the Teacher?	484
8.1	IRL from Failure	485
8.2	Learning Preferences	487
8.3	T-REX: Trajectory-Ranked Reward EXtrapolation	488
8.4	D-REX: Disturbance-Based Reward EXtrapolation	490
9	Let Us Try It Out: IRL for Financial Cliff Walking	490
9.1	Max-Causal Entropy IRL	491
9.2	IRL from Failure	492
9.3	T-REX	493
9.4	Summary	494
10	Financial Applications of IRL	495
10.1	Algorithmic Trading Strategy Identification	495
10.2	Inverse Reinforcement Learning for Option Pricing	497
10.3	IRL of a Portfolio Investor with G-Learning	499
10.4	IRL and Reward Learning for Sentiment-Based Trading Strategies	504
10.5	IRL and the “Invisible Hand” Inference	505
11	Summary	512
12	Exercises	513
	References	515
12	Frontiers of Machine Learning and Finance	519
1	Introduction	519
2	Market Dynamics, IRL, and Physics	521
2.1	“Quantum Equilibrium–Disequilibrium” (QED) Model	522
2.2	The Langevin Equation	523
2.3	The GBM Model as the Langevin Equation	524
2.4	The QED Model as the Langevin Equation	525
2.5	Insights for Financial Modeling	527
2.6	Insights for Machine Learning	528
3	Physics and Machine Learning	529
3.1	Hierarchical Representations in Deep Learning and Physics	529
3.2	Tensor Networks	530
3.3	Bounded-Rational Agents in a Non-equilibrium Environment	534
4	A “Grand Unification” of Machine Learning?	535
4.1	Perception-Action Cycles	537
4.2	Information Theory Meets Reinforcement Learning	538
4.3	Reinforcement Learning Meets Supervised Learning: Predictron, MuZero, and Other New Ideas	539
	References	540
	Index	543

About the Authors

Matthew F. Dixon is an Assistant Professor of Applied Math at the Illinois Institute of Technology. His research in computational methods for finance is funded by Intel. Matthew began his career in structured credit trading at Lehman Brothers in London before pursuing academics and consulting for financial institutions in quantitative trading and risk modeling. He holds a Ph.D. in Applied Mathematics from Imperial College (2007) and has held postdoctoral and visiting professor appointments at Stanford University and UC Davis, respectively. He has published over 20 peer-reviewed publications on machine learning and financial modeling, has been cited in Bloomberg Markets and the Financial Times as an AI in fintech expert, and is a frequently invited speaker in Silicon Valley and on Wall Street. He has published R packages, served as a Google Summer of Code mentor, and is the co-founder of the Thalesians Ltd.

Igor Halperin is a Research Professor in Financial Engineering at NYU and an AI Research Associate at Fidelity Investments. He was previously an Executive Director of Quantitative Research at JPMorgan for nearly 15 years. Igor holds a Ph.D. in Theoretical Physics from Tel Aviv University (1994). Prior to joining the financial industry, he held postdoctoral positions in theoretical physics at the Technion and the University of British Columbia.

Paul Bilokon is CEO and Founder of Thalesians Ltd. and an expert in electronic and algorithmic trading across multiple asset classes, having helped build such businesses at Deutsche Bank and Citigroup. Before focusing on electronic trading, Paul worked on derivatives and has served in quantitative roles at Nomura, Lehman Brothers, and Morgan Stanley. Paul has been educated at Christ Church College, Oxford, and Imperial College. Apart from mathematical and computational finance, his academic interests include machine learning and mathematical logic.

Part I

**Machine Learning with Cross-Sectional
Data**

Chapter 1

Introduction



This chapter introduces the industry context for machine learning in finance, discussing the critical events that have shaped the finance industry's need for machine learning and the unique barriers to adoption. The finance industry has adopted machine learning to varying degrees of sophistication. How it has been adopted is heavily fragmented by the academic disciplines underpinning the applications. We view some key mathematical examples that demonstrate the nature of machine learning and how it is used in practice, with the focus on building intuition for more technical expositions in later chapters. In particular, we begin to address many finance practitioner's concerns that neural networks are a "black-box" by showing how they are related to existing well-established techniques such as linear regression, logistic regression, and autoregressive time series models. Such arguments are developed further in later chapters. This chapter also introduces reinforcement learning for finance and is followed by more in-depth case studies highlighting the design concepts and practical challenges of applying machine learning in practice.

1 Background

In 1955, John McCarthy, then a young Assistant Professor of Mathematics, at Dartmouth College in Hanover, New Hampshire, submitted a proposal with Marvin Minsky, Nathaniel Rochester, and Claude Shannon for the Dartmouth Summer Research Project on Artificial Intelligence (McCarthy et al. 1955). These organizers were joined in the summer of 1956 by Trenchard More, Oliver Selfridge, Herbert Simon, Ray Solomonoff, among others. The stated goal was ambitious:

"The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to

find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves.” Thus the field of artificial intelligence, or AI, was born.

Since this time, AI has perpetually strived to outperform humans on various judgment tasks (Pinar Saygin et al. 2000). The most fundamental metric for this success is the Turing test—a test of a machine’s ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human (Turing 1995). In recent years, a pattern of success in AI has emerged—one in which machines outperform in the presence of a large number of decision variables, usually with the best solution being found through evaluating an exponential number of candidates in a constrained high-dimensional space. Deep learning models, in particular, have proven remarkably successful in a wide field of applications (DeepMind 2016; Kubota 2017; Esteva et al. 2017) including image processing (Simonyan and Zisserman 2014), learning in games (DeepMind 2017), neuroscience (Poggio 2016), energy conservation (DeepMind 2016), skin cancer diagnostics (Kubota 2017; Esteva et al. 2017).

One popular account of this reasoning points to humans’ perceived inability to process large amounts of information and make decisions beyond a few key variables. But this view, even if fractionally representative of the field, does no justice to AI or human learning. Humans are not being replaced any time soon. The median estimate for human intelligence in terms of gigaflops is about 10^4 times more than the machine that ran alpha-go. Of course, this figure is caveated on the important question of whether the human mind is even a Turing machine.

1.1 *Big Data—Big Compute in Finance*

The growth of machine-readable data to record and communicate activities throughout the financial system combined with persistent growth in computing power and storage capacity has significant implications for every corner of financial modeling. Since the financial crises of 2007–2008, regulatory supervisors have reoriented towards “data-driven” regulation, a prominent example of which is the collection and analysis of detailed contractual terms for the bank loan and trading book stress-testing programs in the USA and Europe, instigated by the crisis (Flood et al. 2016).

“Alternative data”—which refers to data and information outside of the usual scope of securities pricing, company fundamentals, or macroeconomic indicators—is playing an increasingly important role for asset managers, traders, and decision makers. Social media is now ranked as one of the top categories of alternative data currently used by hedge funds. Trading firms are hiring experts in machine learning with the ability to apply natural language processing (NLP) to financial news and other unstructured documents such as earnings announcement reports and SEC 10K reports. Data vendors such as Bloomberg, Thomson Reuters, and RavenPack are providing processed news sentiment data tailored for systematic trading models.

In de Prado (2019), some of the properties of these new, alternative datasets are explored: (a) many of these datasets are unstructured, non-numerical, and/or non-categorical, like news articles, voice recordings, or satellite images; (b) they tend to be high-dimensional (e.g., credit card transactions) and the number of variables may greatly exceed the number of observations; (c) such datasets are often sparse, containing NaNs (not-a-numbers); (d) they may implicitly contain information about networks of agents.

Furthermore, de Prado (2019) explains why classical econometric methods fail on such datasets. These methods are often based on linear algebra, which fail when the number of variables exceeds the number of observations. Geometric objects, such as covariance matrices, fail to recognize the topological relationships that characterize networks. On the other hand, machine learning techniques offer the numerical power and functional flexibility needed to identify complex patterns in a high-dimensional space offering a significant improvement over econometric methods.

The “black-box” view of ML is dismissed in de Prado (2019) as a misconception. Recent advances in ML make it applicable to the evaluation of plausibility of scientific theories; determination of the relative informational variables (usually referred to as features in ML) for explanatory and/or predictive purposes; causal inference; and visualization of large, high-dimensional, complex datasets.

Advances in ML remedy the shortcomings of econometric methods in goal setting, outlier detection, feature extraction, regression, and classification when it comes to modern, complex alternative datasets. For example, in the presence of p features there may be up to $2^p - p - 1$ multiplicative interaction effects. For two features there is only one such interaction effect, x_1x_2 . For three features, there are x_1x_2 , x_1x_3 , x_2x_3 , $x_1x_2x_3$. For as few as ten features, there are 1,013 multiplicative interaction effects. Unlike ML algorithms, econometric models do not “learn” the structure of the data. The model specification may easily miss some of the interaction effects. The consequences of missing an interaction effect, e.g. fitting $y_t = x_{1,t} + x_{2,t} + \epsilon_t$ instead of $y_t = x_{1,t} + x_{2,t} + x_{1,t}x_{2,t} + \epsilon_t$, can be dramatic. A machine learning algorithm, such as a decision tree, will recursively partition a dataset with complex patterns into subsets with simple patterns, which can then be fit independently with simple linear specifications. Unlike the classical linear regression, this algorithm “learns” about the existence of the $x_{1,t}x_{2,t}$ effect, yielding much better out-of-sample results.

There is a draw towards more empirically driven modeling in asset pricing research—using ever richer sets of firm characteristics and “factors” to describe and understand differences in expected returns across assets and model the dynamics of the aggregate market equity risk premium (Gu et al. 2018). For example, Harvey et al. (2016) study 316 “factors,” which include firm characteristics and common factors, for describing stock return behavior. Measurement of an asset’s risk premium is fundamentally a problem of prediction—the risk premium is the conditional expectation of a future realized excess return. Methodologies that can reliably attribute excess returns to tradable anomalies are highly prized. Machine learning provides a non-linear empirical approach for modeling realized security

returns from firm characteristics. Dixon and Polson (2019) review the formulation of asset pricing models for measuring asset risk premia and cast neural networks in canonical asset pricing frameworks.

1.2 Fintech

The rise of data and machine learning has led to a “fintech” industry, covering digital innovations and technology-enabled business model innovations in the financial sector (Philippon 2016). Examples of innovations that are central to fintech today include cryptocurrencies and the blockchain, new digital advisory and trading systems, peer-to-peer lending, equity crowdfunding, and mobile payment systems. Behavioral prediction is often a critical aspect of product design and risk management needed for consumer-facing business models; consumers or economic agents are presented with well-defined choices but have unknown economic needs and limitations, and in many cases do not behave in a strictly economically rational fashion. Therefore it is necessary to treat parts of the system as a black-box that operates under rules that cannot be known in advance.

1.2.1 Robo-Advisors

Robo-advisors are financial advisors that provide financial advice or portfolio management services with minimal human intervention. The focus has been on portfolio management rather than on estate and retirement planning, although there are exceptions, such as Blooom. Some limit investors to the ETFs selected by the service, others are more flexible. Examples include Betterment, Wealthfront, Wise-Banyan, FutureAdvisor (working with Fidelity and TD Ameritrade), Blooom, Motif Investing, and Personal Capital. The degree of sophistication and the utilization of machine learning are on the rise among robo-advisors.

1.2.2 Fraud Detection

In 2011 fraud cost the financial industry approximately \$80 billion annually (Consumer Reports, June 2011). According to PwC’s Global Economic Crime Survey 2016, 46% of respondents in the Financial Services industry reported being victims of economic crime in the last 24 months—a small increase from 45% reported in 2014. 16% of those that reported experiencing economic crime had suffered more than 100 incidents, with 6% suffering more than 1,000. According to the survey, the top 5 types of economic crime are asset misappropriation (60%, down from 67% in 2014), cybercrime (49%, up from 39% in 2014), bribery and corruption (18%, down from 20% in 2014), money laundering (24%, as in 2014), and accounting fraud (18%, down from 21% in 2014). Detecting economic crimes is

one of the oldest successful applications of machine learning in the financial services industry. See Gottlieb et al. (2006) for a straightforward overview of some of the classical methods: logistic regression, naïve Bayes, and support vector machines. The rise of electronic trading has led to new kinds of financial fraud and market manipulation. Some exchanges are investigating the use of deep learning to counter spoofing.

1.2.3 Cryptocurrencies

Blockchain technology, first implemented by Satoshi Nakamoto in 2009 as a core component of Bitcoin, is a distributed public ledger recording transactions. Its usage allows secure peer-to-peer communication by linking blocks containing hash pointers to a previous block, a timestamp, and transaction data. Bitcoin is a decentralized digital currency (cryptocurrency) which leverages the blockchain to store transactions in a distributed manner in order to mitigate against flaws in the financial industry.

In contrast to existing financial networks, blockchain based cryptocurrencies expose the entire transaction graph to the public. This openness allows, for example, the most significant agents to be immediately located (pseudonymously) on the network. By processing all financial interactions, we can model the network with a high-fidelity graph, as illustrated in Fig. 1.1 so that it is possible to characterize how the flow of information in the network evolves over time. This novel data representation permits a new form of financial econometrics—with the emphasis on the topological network structures in the microstructure rather than solely the covariance of historical time series of prices. The role of users, entities, and their interactions in formation and dynamics of cryptocurrency risk investment, financial predictive analytics and, more generally, in re-shaping the modern financial world is a novel area of research (Dyhrberg 2016; Gomber et al. 2017; Sovbetov 2018).

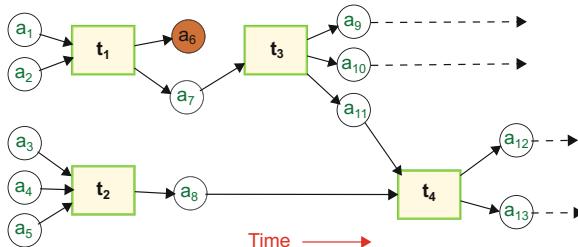


Fig. 1.1 A transaction–address graph representation of the Bitcoin network. Addresses are represented by circles, transactions with rectangles, and edges indicate a transfer of coins. Blocks order transactions in time, whereas each transaction with its input and output nodes represents an immutable decision that is encoded as a subgraph on the Bitcoin network. Source: Akcora et al. (2018)

2 Machine Learning and Prediction

With each passing year, finance becomes increasingly reliant on computational methods. At the same time, the growth of machine-readable data to monitor, record, and communicate activities throughout the financial system has significant implications for how we approach the topic of modeling. One of the reasons that AI and the set of computer algorithms for learning, referred to as “machine learning,” have been successful is a result of a number of factors beyond computer hardware and software advances. Machines are able to model complex and high-dimensional data generation processes, sweep through millions of model configurations, and then robustly evaluate and correct the models in response to new information (Dhar 2013). By continuously updating and hosting a number of competing models, they prevent any one model leading us into a data gathering silo effective only for that market view. Structurally, the adoption of ML has even shifted our behavior—the way we reason, experiment, and shape our perspectives from data using ML has led to empirically driven trading and investment decision processes.

Machine learning is a broad area, covering various classes of algorithms for pattern recognition and decision-making. In **supervised learning**, we are given labeled data, i.e. pairs $(x_1, y_1), \dots, (x_n, y_n)$, $x_1, \dots, x_n \in X$, $y_1, \dots, y_n \in Y$, and the goal is to learn the relationship between X and Y . Each observation x_i is referred to as a **feature vector** and y_i is the **label or response**. In **unsupervised learning**, we are given unlabeled data, x_1, x_2, \dots, x_n and our goal is to retrieve exploratory information about the data, perhaps grouping similar observations or capturing some hidden patterns. Unsupervised learning includes **cluster analysis** algorithms such as hierarchical clustering, k -means clustering, self-organizing maps, Gaussian mixture, and hidden Markov models and is commonly referred to as data mining. In both instances, the data could be financial time series, news documents, SEC documents, and textual information on important events. The third type of machine learning paradigm is **reinforcement learning** and is an algorithmic approach for enforcing Bellman optimality of a Markov Decision Process—defining a set of states and actions in response to a changing regime so as to maximize some notion of cumulative reward. In contrast to supervised learning, which just considers a single action at each point in time, reinforcement learning is concerned with the optimal sequence of actions. It is therefore a form of dynamic programming that is used for decisions leading to optimal trade execution, portfolio allocation, and liquidation over a given horizon.

Supervised learning addresses a fundamental prediction problem: Construct a non-linear predictor, $\hat{Y}(X)$, of an output, Y , given a high-dimensional input matrix $X = (X_1, \dots, X_P)$ of P variables. Machine learning can be simply viewed as the study and construction of an input–output map of the form

$$Y = F(X) \text{ where } X = (X_1, \dots, X_P).$$

$F(X)$ is sometimes referred to as the “data-feature” map. The output variable, Y , can be continuous, discrete, or mixed. For example, in a classification problem,

$F : X \rightarrow G$, where $G \in \mathcal{K} := \{0, \dots, K - 1\}$, K is the number of categories and \hat{G} is the predictor.

Supervised machine learning uses a parameterized¹ model $g(X|\theta)$ over independent variables X , to predict the continuous or categorical output Y or G . The model is parameterized by one or more free parameters θ which are fitted to data. Prediction of categorical variables is referred to as *classification* and is common in pattern recognition. The most common approach to predicting categorical variables is to encode the response G as one or more binary values, then treat the model prediction as continuous.

? Multiple Choice Question 1

Select all the following correct statements:

1. Supervised learning involves learning the relationship between input and output variables.
2. Supervised learning requires a human supervisor to prepare labeled training data.
3. Unsupervised learning does not require a human supervisor and is therefore superior to supervised learning.
4. Reinforcement learning can be viewed as a generalization of supervised learning to Markov Decision Processes.

There are two different classes of supervised learning models, *discriminative* and *generative*. A discriminative model learns the decision boundary between the classes and implicitly learns the distribution of the output conditional on the input. A generative model explicitly learns the joint distribution of the input and output. An example of the former is a neural network or a decision tree and a restricted Boltzmann machine (RBM) is an example of the latter. Learning the joint distribution has the advantage that by the Bayes' rule, it can also give the conditional distribution of the output given the input, but also be used for other purposes such as selecting features based on the joint probability. Generative models are typically more difficult to build.

This book will mostly focus on discriminative models only, but the distinction should be made clear. A discriminative model predicts the probability of an output given an input. For example, if we are predicting the probability of a label $G = k$, $k \in \mathcal{K}$, then $g(x|\theta)$ is a map $g : \mathbb{R}^p \rightarrow [0, 1]^K$ and the outputs represent a discrete probability distribution over G referred to as a “one-hot” encoding—a K -vector of zeros with 1 at the k th position:

$$\hat{G}_k := \mathbb{P}(G = k \mid X = x, \theta) = g_k(x|\theta) \quad (1.1)$$

¹The model is referred to as *non-parametric* if the parameter space is infinite dimensional and *parametric* if the parameter space is finite dimensional.

and hence we have that

$$\sum_{k \in \mathcal{K}} g_k(\mathbf{x}|\boldsymbol{\theta}) = 1. \quad (1.2)$$

In particular, when G is dichotomous ($K = 2$), the second component of the model output is the conditional expected value of G

$$\hat{G} := \hat{G}_1(\mathbf{x}|\boldsymbol{\theta}) = 0 \cdot \mathbb{P}(G = 0 | X = \mathbf{x}, \boldsymbol{\theta}) + 1 \cdot \mathbb{P}(G = 1 | X = \mathbf{x}, \boldsymbol{\theta}) = \mathbb{E}[G | X = \mathbf{x}, \boldsymbol{\theta}]. \quad (1.3)$$

The conditional variance of G is given by

$$\sigma^2 := \mathbb{E}[(G - \hat{G})^2 | X = \mathbf{x}, \boldsymbol{\theta}] = g_1(\mathbf{x}|\boldsymbol{\theta}) - (g_1(\mathbf{x}|\boldsymbol{\theta}))^2, \quad (1.4)$$

which is an inverted parabola with a maximum at $g_1(\mathbf{x}|\boldsymbol{\theta}) = 0.5$. The following example illustrates a simple discriminative model which, here, is just based on a set of fixed rules for partitioning the input space.

Example 1.1 Model Selection

Suppose $G \in \{A, B, C\}$ and the input $X \in \{0, 1\}^2$ are binary 2-vectors given in Table 1.1.

Table 1.1 Sample model data

G	\mathbf{x}
A	(0, 1)
B	(1, 1)
C	(1, 0)
C	(0, 0)

To match the input and output in this case, one could define a parameter-free step function $g(\mathbf{x})$ over $\{0, 1\}^2$ so that

$$g(\mathbf{x}) = \begin{cases} \{1, 0, 0\} & \text{if } \mathbf{x} = (0, 1) \\ \{0, 1, 0\} & \text{if } \mathbf{x} = (1, 1) \\ \{0, 0, 1\} & \text{if } \mathbf{x} = (1, 0) \\ \{0, 0, 1\} & \text{if } \mathbf{x} = (0, 0). \end{cases} \quad (1.5)$$

The discriminative model $g(\mathbf{x})$, defined in Eq. 1.5, specifies a set of fixed rules which predict the outcome of this experiment with 100% accuracy. Intuitively, it seems clear that such a model is flawed if the actual relation between inputs and outputs is non-deterministic. Clearly, a skilled analyst would typically not build such

a model. Yet, hard-wired rules such as this are ubiquitous in the finance industry such as rule-based technical analysis and heuristics used for scoring such as credit ratings.

If the model is allowed to be general, there is no reason why this particular function should be excluded. Therefore automated systems analyzing datasets such as this may be prone to construct functions like those given in Eq. 1.5 unless measures are taken to prevent it. It is therefore incumbent on the model designer to understand what makes the rules in Eq. 1.5 objectionable, with the goal of using a theoretically sound process to generalize the input–output map to other data.

Example 1.2 Model Selection (Continued)

Consider an alternate model for Table 1.1

$$h(\mathbf{x}) = \begin{cases} \{0.9, 0.05, 0.05\} & \text{if } \mathbf{x} = (0, 1) \\ \{0.05, 0.9, 0.05\} & \text{if } \mathbf{x} = (1, 1) \\ \{0.05, 0.05, 0.9\} & \text{if } \mathbf{x} = (1, 0) \\ \{0.05, 0.05, 0.9\} & \text{if } \mathbf{x} = (0, 0). \end{cases}$$

If this model were sampled, it would produce the data in Table 1.1 with probability $(0.9)^4 = 0.6561$. We can hardly exclude this model from consideration on the basis of the results in Table 1.1, so which one do we choose?

Informally, the heart of the model selection problem is that model g has excessively high confidence about the data, when that confidence is often not warranted. Many other functions, such as h , could have easily generated Table 1.1. Though there is only one model that can produce Table 1.1 with probability 1.0, there is a whole family of models that can produce the table with probability at least 0.66. Many of these plausible models do not assign overwhelming confidence to the results. To determine which model is best on average, we need to introduce another key concept.

2.1 Entropy

Model selection in machine learning is based on a quantity known as **entropy**. Entropy represents the amount of information associated with each event. To illustrate the concept of entropy, let us consider a non-fair coin toss. There are two outcomes, $\Omega = \{H, T\}$. Let Y be a Bernoulli random variable representing the coin flip with density $f(Y = 1) = \mathbb{P}(H) = p$ and $f(Y = 0) = \mathbb{P}(T) = 1 - p$. The (binary) entropy of Y under f is

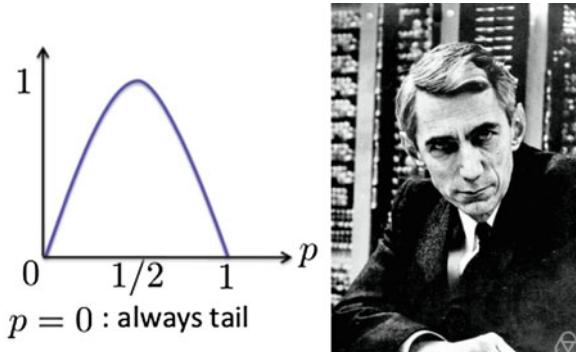


Fig. 1.2 (Left) This figure shows the binary entropy of a biased coin. If the coin is fully biased, then each flip provides no new information as the outcome is already known and hence the entropy is zero. (Right) The concept of entropy was introduced by Claude Shannon² in 1948³ and was originally intended to represent an upper limit on the average length of a lossless compression encoding. Shannon's entropy is foundational to the mathematical discipline of information theory

$$\mathcal{H}(f) = -p \log_2 p - (1-p)\log_2(1-p) \leq 1\text{bit}. \quad (1.6)$$

The reason why base 2 is chosen is so that the upper bound represents the number of bits needed to represent the outcome of the random variable, i.e. $\{0, 1\}$ and hence 1 bit.

The binary entropy for a biased coin is shown in Fig. 1.2. If the coin is fully biased, then each flip provides no new information as the outcome is already known. The maximum amount of information that can be revealed by a coin flip is when the coin is unbiased.

Let us now reintroduce our parameterized mass in the setting of the biased coin. Let us consider an i.i.d. discrete random variable $Y : \Omega \rightarrow \mathcal{Y} \subset \mathbb{R}$ and let

$$g(y|\theta) = \mathbb{P}(\omega \in \Omega; Y(\omega) = y)$$

denote a parameterized probability mass function for Y .

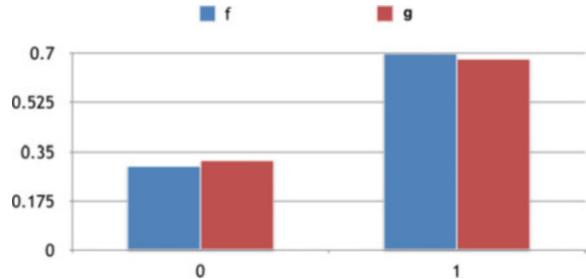
We can measure how different $g(y|\theta)$ is from the true density $f(y)$ using the cross-entropy

$$\mathcal{H}(f, g) := -\mathbb{E}_f [\log_2 g] = \sum_{y \in \mathcal{Y}} f(y) \log_2 g(y|\theta) \geq \mathcal{H}(f), \quad (1.7)$$

²Photo: Jacobs, Konrad [CC BY-SA 2.0 de (<https://creativecommons.org/licenses/by-sa/2.0/deed.en>)].

³C. Shannon, A Mathematical Theory of Communication, The Bell System Technical Journal, Vol. 27, pp. 379-423, 623-656, July, October, 1948.

Fig. 1.3 A comparison of the true distribution, f , of a biased coin with a parameterized model g of the coin



so that $\mathcal{H}(f, f) = \mathcal{H}(f)$, where $\mathcal{H}(f)$ is the entropy of f :

$$\mathcal{H}(f) := -\mathbb{E}_f[\log_2 f] = -\sum_{y \in \mathcal{Y}} f(y) \log_2 f(y). \quad (1.8)$$

If $g(y|\theta)$ is a model of the non-fair coin with $g(Y=1|\theta) = p_\theta$, $g(Y=0|\theta) = 1 - p_\theta$. The cross-entropy is

$$\mathcal{H}(f, g) = -p \log_2 p_\theta - (1-p) \log_2 (1-p_\theta) \geq -p \log_2 p - (1-p) \log_2 (1-p). \quad (1.9)$$

Let us suppose that $p = 0.7$ and $p_\theta = 0.68$, as illustrated in Fig. 1.3, then the cross-entropy is

$$\mathcal{H}(f, g) = -0.3 \log_2(0.32) - 0.7 \log_2(0.68) = 0.8826322.$$

Returning to our experiment in Table 1.1, let us consider the cross-entropy of these models which, as you will recall, depends on inputs too. Model g completely characterizes the data in Table 1.1 and we interpret it here as the truth. Model h , however, only summarizes some salient aspects of the data, and there is a large family of tables that would be consistent with model h . In the presence of noise or strong evidence indicating that Table 1.1 was the only possible outcome, we should interpret models like h as a more plausible explanation of the actual underlying phenomenon.

Evaluating the cross-entropy between model h and model g , we get $-\log_2(0.9)$ for each observation in the table, which gives the negative log-likelihood when summed over all samples. The cross-entropy is at its minimum when $h = g$, we get $-\log_2(1.0) = 0$. If g were a parameterized model, then clearly minimizing cross-entropy or equivalently maximizing log-likelihood gives the maximum likelihood estimate of the parameter. We shall revisit the topic of parameter estimation in Chap. 2.

? Multiple Choice Question 2

Select all of the following statements that are correct:

1. Neural network classifiers are a discriminative model which output probabilistic weightings for each category, given an input feature vector.
 2. If the data is independent and identically distributed (i.i.d.), then the output of a dichotomous classifier is a conditional probability of a Bernoulli random variable.
 3. A θ -parameterized discriminative model for a biased coin dependent on the environment X can be written as $\{g_i(X|\theta)\}_{i=0}^1$.
 4. A model of two biased coins, both dependent on the environment X , can be equivalently modeled with either the pair $\{g_i^{(1)}(X|\theta)\}_{i=0}^1$ and $\{g_i^{(2)}(X|\theta)\}_{i=0}^1$, or the multi-classifier $\{g_i(X|\theta)\}_{i=0}^3$.
-

2.2 Neural Networks

Neural networks represent the non-linear map $F(X)$ over a high-dimensional input space using hierarchical layers of abstractions. An example of a neural network is a feedforward network—a sequence of L layers⁴ formed via composition:

> Deep Feedforward Networks

A deep feedforward network is a function of the form

$$\hat{Y}(X) := F_{W,b}(X) = \left(f_{W^{(L)}, b^{(L)}}^{(L)} \circ \dots \circ f_{W^{(1)}, b^{(1)}}^{(1)} \right)(X),$$

where

- $f_{W^{(l)}, b^{(l)}}^{(l)}(X) := \sigma^{(l)}(W^{(l)}X + b^{(l)})$ is a semi-affine function, where $\sigma^{(l)}$ is a univariate and continuous non-linear activation function such as $\max(\cdot, 0)$ or $\tanh(\cdot)$.
- $W = (W^{(1)}, \dots, W^{(L)})$ and $b = (b^{(1)}, \dots, b^{(L)})$ are weight matrices and offsets (a.k.a. biases), respectively.

⁴Note that we do not treat the input as a layer. So there are $L - 1$ hidden layers and an output layer.

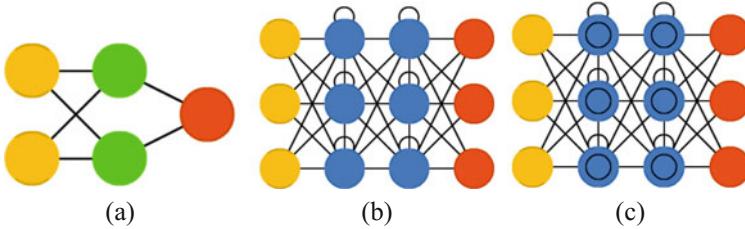


Fig. 1.4 Examples of neural networks architectures discussed in this book. Source: Van Veen, F. & Leijnen, S. (2019), “The Neural Network Zoo,” Retrieved from <https://www.asimovinstitute.org/neural-network-zoo>. The input nodes are shown in yellow and represent the input variables, the green nodes are the hidden neurons and present hidden latent variables, the red nodes are the outputs or responses. Blue nodes denote hidden nodes with recurrence or memory. (a) Feedforward. (b) Recurrent. (c) Long short-term memory

An earlier example of a feedforward network architecture is given in Fig. 1.4a. The input nodes are shown in yellow and represent the input variables, the green nodes are the hidden neurons and present hidden latent variables, the red nodes are the outputs or responses. The activation functions are essential for the network to approximate non-linear functions. For example, if there is one hidden layer and $\sigma^{(1)}$ is the identity function, then

$$\hat{Y}(X) = W^{(2)}(W^{(1)}X + b^{(1)}) + b^{(2)} = W^{(2)}W^{(1)}X + W^{(2)}b^{(1)} + b^{(2)} = W'X + b' \quad (1.10)$$

is just linear regression, i.e. an affine transformation.⁵ Clearly, if there are no hidden layers, the architecture recovers standard linear regression

$$Y = WX + b$$

and logistic regression $\phi(WX + b)$, where ϕ is a sigmoid or softmax function, when the response is continuous or categorical, respectively. Some of the terminology used here and the details of this model will be described in Chap. 4.

The theoretical roots of feedforward neural networks are given by the Kolmogorov–Arnold representation theorem (Arnold 1957; Kolmogorov 1957) of multivariate functions. Remarkably, Hornik et al. (1989) showed how neural networks, with one hidden layer, are universal approximators to non-linear functions.

Clearly there are a number of issues in any architecture design and inference of the model parameters (W, b). How many layers? How many neurons N_l in each hidden layer? How to perform “variable selection”? How to avoid over-fitting? The details and considerations given to these important questions will be addressed in Chap. 4.

⁵While the functional form of the map is the same as linear regression, neural networks do not assume a data generation process and hence inference is not identical to ordinary least squares regression.

3 Statistical Modeling vs. Machine Learning

Supervised machine learning is often an algorithmic form of statistical model estimation in which the data generation process is treated as an unknown (Breiman 2001). Model selection and inference is automated, with an emphasis on processing large amounts of data to develop robust models. It can be viewed as a highly efficient data compression technique designed to provide predictors in complex settings where relations between input and output variables are non-linear and input space is often high-dimensional. Machine learners balance filtering data with the goal of making accurate and robust decisions, often discrete and as a categorical function of input data.

This fundamentally differs from maximum likelihood estimators used in standard statistical models, which assume that the data was generated by the model and typically have difficulty with over-fitting, especially when applied to high-dimensional datasets. Given the complexity of modern datasets, whether they are limit order books or high-dimensional financial time series, it is increasingly questionable whether we can posit inference on the basis of a known data generation process. It is a reasonable assertion, even if an economic interpretation of the data generation process can be given, that the exact form cannot be known all the time.

The paradigm that machine learning provides for data analysis therefore is very different from the traditional statistical modeling and testing framework. Traditional fit metrics, such as R^2 , t -values, p -values, and the notion of *statistical significance*, are replaced by out-of-sample forecasting and understanding the bias–variance tradeoff. Machine learning is data-driven and focuses on finding structure in large datasets. The main tools for variable or predictor selection are *regularization* and *dropout* which are discussed in detail in Chap. 4.

Table 1.2 contrasts maximum likelihood estimation-based inference with supervised machine learning. The comparison is somewhat exaggerated for ease of explanation. Rather the two approaches should be viewed as opposite ends of a continuum of methods. Linear regression techniques such as LASSO and ridge regression, or hybrids such as Elastic Net, fall somewhere in the middle, providing some combination of the explanatory power of maximum likelihood estimation while retaining out-of-sample predictive performance on high-dimensional datasets.

3.1 Modeling Paradigms

Machine learning and statistical methods can be further characterized by whether they are parametric or non-parametric. *Parametric models* assume some finite set of parameters and attempt to model the response as a function of the input variables and the parameters. Due to the finiteness of the parameter space, they have limited flexibility and cannot capture complex patterns in big data. As a general rule, examples of parametric models include ordinary least squares linear

Table 1.2 This table contrasts maximum likelihood estimation-based inference with supervised machine learning. The comparison is somewhat exaggerated for ease of explanation; however, the two should be viewed as opposite ends of a continuum of methods. Regularized linear regression techniques such as LASSO and ridge regression, or hybrids such as Elastic Net, provide some combination of the explanatory power of maximum likelihood estimation while retaining out-of-sample predictive performance on high-dimensional datasets

Property	Statistical inference	Supervised machine learning
Goal	Causal models with explanatory power	Prediction performance, often with limited explanatory power
Data	The data is generated by a model	The data generation process is unknown
Framework	Probabilistic	Algorithmic and Probabilistic
Expressibility	Typically linear	Non-linear
Model selection	Based on information criteria	Numerical optimization
Scalability	Limited to lower-dimensional data	Scales to high-dimensional input data
Robustness	Prone to over-fitting	Designed for out-of-sample performance
Diagnostics	Extensive	Limited

regression, polynomial regression, mixture models, neural networks, and hidden Markov models.

Non-parametric models treat the parameter space as infinite dimensional—this is equivalent to introducing a hidden or latent function. The model structure is, for the most part, not specified a priori and they can grow in complexity with more data. Examples of non-parametric models include kernel methods such as support vector machines and Gaussian processes, the latter will be the focus of Chap. 3.

Note that there is a gray area in whether neural networks are parametric or non-parametric and it strictly depends on how they are fitted. For example, it is possible to treat the parameter space in a neural network as infinite dimensional and hence characterize neural networks as non-parametric (see, for example, Philipp and Carbonell (2017)). However, this is an exception rather than the norm.

While on the topic of modeling paradigms, it is helpful to further distinguish between *probabilistic models*, the subject of the next two chapters, and deterministic models, the subject of Chaps. 4, 5, and 8. The former treats the parameters as random and the latter assumes that the parameters are given.

Within probabilistic modeling, a particular niche is occupied by the so-called *state-space models*. In these models one assumes the existence of a certain unobserved, latent, process, whose evolution drives a certain observable process. The evolution of the latent process and the dependence of the observable process on the latent process may be given in stochastic, probabilistic terms, which places the state-space models within the realm of probabilistic modeling.

Note, somewhat counter to the terminology, that a deterministic model may produce a probabilistic output, for example, a logistic regression gives the probability that the response is positive given the input variables. The choice of whether to use a probabilistic or deterministic model is discussed further in the next chapter

and falls under the more general and divisive topic of “Bayesian versus frequentist modeling.”

3.2 Financial Econometrics and Machine Learning

Machine learning generalizes parametric methods in financial econometrics. A taxonomy of machine learning in econometrics is shown in Fig. 1.5 together with the section references to the material in the first two parts of this book.

When the data is a time series, neural networks can be configured with recurrence to build memory into the model. By relaxing the modeling assumptions needed for econometrics techniques, such as ARIMA (Box et al. 1994) and GARCH models (Bollerslev 1986), recurrent neural networks provide a semi-parametric or even non-parametric extension of classical time series methods. That use, however, comes with much caution. Whereas financial econometrics is built on rigorous experimental design methods such as the estimation framework of Box and Jenkins (1976), recurrent neural networks have grown from the computational engineering literature and many engineering studies overlook essential diagnostics such as Dickey–Fuller tests for verifying stationarity of the time series, a critical aspect of financial time series modeling. We take an integrative approach, showing how to cast recurrent neural networks into financial econometrics frameworks such as Box-Jenkins.

More formally, if the input–output pairs $\mathcal{D} = \{X_t, Y_t\}_{t=1}^N$ are autocorrelated observations of X and Y at times $t = 1, \dots, N$, then the fundamental prediction problem can be expressed as a sequence prediction problem: construct a non-linear

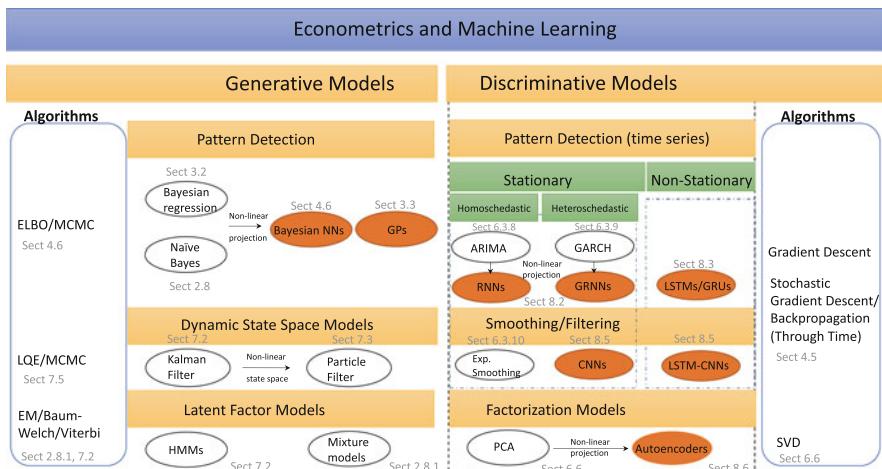


Fig. 1.5 Overview of how machine learning generalizes parametric econometrics, together with the section references to the material in the first two parts of this book

times series predictor, $\hat{Y}(X_t)$, of an output, Y , using a high-dimensional input matrix of T length sub-sequences X_t :

$$\hat{Y}_t = F(X_t) \text{ where } X_t := seq_{T,0}(X_t) := (X_{t-T+1}, \dots, X_t), \quad (1.11)$$

where X_{t-j} is a j th lagged observation of X_t , $X_{t-j} = L^j[X_j]$, for $0 = 1, \dots, T - 1$. Sequence learning, then, is just a composition of a non-linear map and a vectorization of the lagged input variables. If the data is i.i.d., then no sequence is needed (i.e., $T = 1$), and we recover the standard cross-sectional prediction problem which can be approximated with a feedforward neural network model.

Recurrent neural networks (RNNs), shown in Fig. 1.4b, are time series methods or sequence learners which have achieved much success in applications such as natural language understanding, language generation, video processing, and many other tasks (Graves 2012). There are many types of RNNs—we will just concentrate on simple RNN models for brevity of notation. Like multivariate structural autoregressive models, RNNs apply an autoregressive function $f_{W^{(1)}, b^{(1)}}^{(1)}(X_t)$ to each input sequence X_t , where T denotes the look back period at each time step—the maximum number of lags. However, rather than directly imposing a linear autocovariance structure, a RNN provides a flexible functional form to directly model the predictor, \hat{Y} .

A simple RNN can be understood as an unfolding of a single hidden layer neural network (a.k.a. Elman network (Elman 1991)) over all time steps in the sequence, $j = 0, \dots, T$. For each time step, j , this function $f_{W^{(1)}, b^{(1)}}^{(1)}(X_{t,j})$ generates a hidden state Z_{t-j} from the current input X_{t-j} and the previous hidden state Z_{t-j-1} and $X_{t,j} = seq_{T,j}(X_t) \subset X_t$ which appears in general form as:

$$\text{response: } \hat{Y}_t = f_{W^{(2)}, b^{(2)}}^{(2)}(Z_t) := \sigma^{(2)}(W^{(2)}Z_t + b^{(2)}),$$

$$\begin{aligned} \text{hidden states: } Z_{t-j} &= f_{W^{(1)}, b^{(1)}}^{(1)}(X_{t,j}) \\ &:= \sigma^{(1)}(W_z^{(1)}Z_{t-j-1} + W_x^{(1)}X_{t-j} + b^{(1)}), \quad j \in \{T, \dots, 0\}, \end{aligned}$$

where $\sigma^{(1)}$ is an activation function such as $\tanh(x)$ and $\sigma^{(2)}$ is either a softmax function, sigmoid function, or identity function depending on whether the response is categorical, binary, or continuous, respectively. The connections between the extremal inputs X_t and the H hidden units are weighted by the time invariant matrix $W_x^{(1)} \in \mathbb{R}^{H \times P}$. The recurrent connections between the H hidden units are weighted by the time invariant matrix $W_z^{(1)} \in \mathbb{R}^{H \times H}$. Without such a matrix, the architecture is simply a single layered feedforward network without memory—each independent observation X_t is mapped to an output \hat{Y}_t using the same hidden layer.

It is important to note that a plain RNN, de-facto, is not a deep network. The recurrent layer has the deceptive appearance of being a deep network when “unfolded,” i.e. viewed as being repeatedly applied to each new input, X_{t-j} , so that $Z_{t-j} = \sigma^{(1)}(W_z^{(1)}Z_{t-j-1} + W_x^{(1)}X_{t-j})$. However the same recurrent weights

remain fixed over all repetitions—there is only one recurrent layer with weights $W_z^{(1)}$.

The amount of memory in the model is equal to the sequence length T . This means that the maximum lagged input that affects the output, \hat{Y}_t , is X_{t-T} . We shall see later in Chap. 8 that RNNs are simply non-linear autoregressive models with exogenous variables (NARX). In the special case of the univariate time series prediction $\hat{X}_t = F(X_{t-1})$, using $T = p$ previous observations $\{X_{t-i}\}_{i=1}^p$, only one neuron in the recurrent layer with weight ϕ and no activation function, a RNN is an AR(p) model with zero drift and geometric weights:

$$\hat{X}_t = (\phi_1 L + \phi_2 L^2 + \cdots + \phi_p L^p)[X_t], \quad \phi_i := \phi^i,$$

with $|\phi| < 1$ to ensure that the model is stationary. The order p can be found through autocorrelation tests of the residual if we make the additional assumption that the error $X_t - \hat{X}_t$ is Gaussian. Example tests include the Ljung–Box and Lagrange multiplier tests. However, the over-reliance on parametric diagnostic tests should be used with caution since the conditions for satisfying the tests may not be satisfied on complex time series data. Because the weights are time independent, plain RNNs are static time series models and not suited to non-covariance stationary time series data.

Additional layers can be added to create deep RNNs by stacking them on top of each other, using the hidden state of the RNN as the input to the next layer. However, RNNs have difficulty in learning long-term dynamics, due in part to the vanishing and exploding gradients that can result from propagating the gradients down through the many unfolded layers of the network. Moreover, RNNs like most methods in supervised machine learning are inherently designed for stationary data. Oftentimes, financial time series data is non-stationary.

In Chap. 8, we shall introduce gated recurrent units (GRUs) and long short term memory (LSTM) networks, the latter is shown in Fig. 1.4c as a particular form of recurrent network which provide a solution to this problem by incorporating memory units. In the language of time series modeling, we shall construct dynamic RNNs which are suitable for non-stationary data. More precisely, we shall see that these architecture shall learn when to forget previous hidden states and when to update hidden states given new information.

This ability to model hidden states is of central importance in financial time series modeling and applications in trading. Mixture models and hidden Markov models have historically been the primary probabilistic methods used in quantitative finance and econometrics to model regimes and are reviewed in Chap. 2 and Chap. 7 respectively. Readers are encouraged to review Chap. 2, before reading Chap. 7.

? Multiple Choice Question 3

Select all the following correct statements:

1. A linear recurrent neural network with a memory of p lags is an autoregressive model $AR(p)$ with non-parametric error.

2. Recurrent neural networks, as time series models, are guaranteed to be stationary, for any choice of weights.
 3. The amount of memory in a shallow recurrent network corresponds to the number of times a single perceptron layer is unfolded.
 4. The amount of memory in a deep recurrent network corresponds to the number of perceptron layers.
-

3.3 Over-fitting

Undoubtedly the pivotal concern with machine learning, and especially deep learning, is the propensity for over-fitting given the number of parameters in the model. This is why skill is needed to fit deep neural networks.

In frequentist statistics, over-fitting is addressed by penalizing the likelihood function with a penalty term. A common approach is to select models based on Akaike's information criteria (Akaike 1973), which assumes that the model error is Gaussian. The penalty term is in fact a sample bias correction term to the Kullback–Leibler divergence (the relative Entropy) and is applied post-hoc to the unpenalized maximized loss likelihood.

Machine learning methods such as least absolute shrinkage and selection operator (LASSO) and ridge regression more conveniently directly optimize a loss function with a penalty term. Moreover the approach is not restricted to modeling error distributional assumptions. LASSO or L_1 regularization favors sparser parameterizations, whereas ridge regression or L_2 reduces the magnitude of the parameters. Regularization is arguably the most important aspect of why machine learning methods have been so successful in finance and other distributions. Conversely, its absence is why neural networks fell out-of-favor in the finance industry in the 1990s.

Regularization and information criteria are closely related, a key observation which enables us to express model selection in terms of information entropy and hence root our discourse in the works of Shannon (1948), Wiener (1964), Kullback and Leibler (1951). How to choose weights, the concept of regularization for model selection, and cross-validation is discussed in Chap. 4.

It turns out that the choice of priors in Bayesian modeling provides a probabilistic analog to LASSO and ridge regression. L_2 regularization is equivalent to a Gaussian prior and L_1 is an equivalent to a Laplacian prior. Another important feature of Bayesian models is that they have a natural mechanism for prevention of over-fitting built-in. Introductory Bayesian modeling is covered extensively in Chap. 2.

4 Reinforcement Learning

Recall that supervised learning is essentially a paradigm for inferring the parameters of a map between input data and an output through minimizing an error over training samples. Performance generalization is achieved through estimating regularization parameters on cross-validation data. Once the weights of a network are learned, they are not updated in response to new data. For this reason, supervised learning can be considered as an “offline” form of learning, i.e. the model is fitted offline. Note that we avoid referring to the model as static since it is possible, under certain types of architectures, to create a dynamical model in which the map between input and output changes over time. For example, as we shall see in Chap. 8, a LSTM maintains a set of hidden state variables which result in a different form of the map over time.

In such learning, a “teacher” provides an exact right output for each data point in a training set. This can be viewed as “feedback” from the teacher, which for supervised learning amounts to informing the agent with the correct label each time the agent classifies a new data point in the training dataset. Note that this is opposite to unsupervised learning, where there is no teacher to provide correct answers to a ML algorithm, which can be viewed as a setting with no teacher, and, respectively, no feedback from a teacher.

An alternative learning paradigm, referred to as “reinforcement learning,” exists which models a sequence of decisions over state space. The key difference of this setting from supervised learning is feedback from the teacher is somewhat in between of the two extremes of unsupervised learning (no feedback at all) and supervised learning that can be viewed as feedback by providing the right labels. Instead, such partial feedback is provided by “rewards” which encourage a desired behavior, but without explicitly instructing the agent what exactly it should do, as in supervised learning.

The simplest way to reason about reinforcement learning is to consider machine learning tasks as a problem of an agent interacting with an environment, as illustrated in Fig. 1.6.

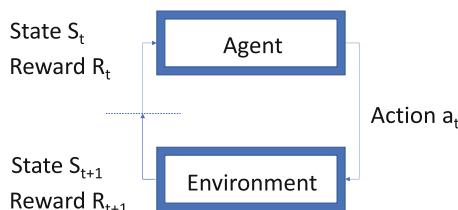


Fig. 1.6 This figure shows a reinforcement learning agent which performs actions at times t_0, \dots, t_n . The agent perceives the environment through the state variable S_t . In order to perform better on its task, feedback on an action a_t is provided to the agent at the next time step in the form of a reward R_t

The agent learns about the environment in order to perform better on its task, which can be formulated as the problem of performing an optimal **action**. If an action performed by an agent is always the same and does not impact the environment, in this case we simply have a perception task, because learning about the environment helps to improve performance on this task. For example, you might have a model for prediction of mortgage defaults where the action is to compute the default probability for a given mortgage. The agent, in this case, is just a predictive model that produces a number and there is measurement of how the model impacts the environment. For example, if a model at a large mortgage broker predicted that all borrowers will default, it is very likely that this would have an impact on the mortgage market, and consequently future predictions. However, this feedback is ignored as the agent just performs perception tasks, ideally suited for supervised learning. Another example is in trading. Once an action is taken by the strategy there is feedback from the market which is referred to as “market impact.”

Such a learner is configured to maximize a long-run utility function under some assumptions about the environment. One simple assumption is to treat the environment as being fully observable and evolving as a first-order Markov process. A Markov Decision Process (MDP) is then the simplest modeling framework that allows us to formalize the problem of reinforcement learning. A task solved by MDPs is the problem of **optimal control**, which is the problem of choosing action variables over some period of time, in order to maximize some objective function that depends both on the future states and action taken. In a discrete-time setting, the state of the environment $S_t \in \mathcal{S}$ is used by the learner (a.k.a. agent) to decide which action $a_t \in \mathcal{A}(S_t)$ to take at each time step. This decision is made dynamic by updating the probabilities of selecting each action conditioned on S_t . These conditional probabilities $\pi_t(a|s)$ are referred to as the agent’s **policy**. The mechanism for updating the policy as a result of its learning is as follows: one time step later and as a consequence of its action, the learner receives a reward defined by a **reward function**, an immediate reward given the current state S_t and action taken a_t .

As a result of the dynamic environment and the action of the agent, we transition to a new state S_{t+1} . A reinforcement learning method specifies how to change the policy so as to maximize the total amount of reward received over the long-run. The constructs for reinforcement learning will be formalized in Chap. 9 but we shall informally discuss some of the challenges of reinforcement learning in finance here.

Most of the impressive progress reported recently with reinforcement learning by researchers and companies such as Google’s DeepMind or OpenAI, such as playing video games, walking robots, self-driving cars, etc., assumes complete observability, using Markovian dynamics. The much more challenging problem, which is a better setting for finance, is how to formulate reinforcement learning for partially observable environments, where one or more variables are hidden.

Another, more modest, challenge exists in how to choose the optimal policy when no environment is fully observable but the dynamic process for how the states evolve over time is unknown. It may be possible, for simple problems, to reason about how the states evolve, perhaps adding constraints on the state-action space. However,

the problem is especially acute in high-dimensional discrete state spaces, arising from, say, discretizing continuous state spaces. Here, it is typically intractable to enumerate all combinations of states and actions and it is hence not possible to exactly solve the optimal control problem. Chapter 9 will present approaches for approximating the optimal control problem. In particular, we will turn to neural networks to approximate an action function known as a “Q-function.” Such an approach is referred to as “Q-Learning” and more recently, with the use of deep learning to approximate the Q-function, is referred to as “Deep Q-Learning.”

To fix ideas, we consider a number of examples to illustrate different aspects of the problem formulation and challenge in applying reinforcement learning. We start with arguably the most famous toy problem used to study stochastic optimal control theory, the “multi-armed bandit problem.” This problem is especially helpful in developing our intuition of how an agent must balance the competing goals of exploring different actions versus exploitation of known outcomes.

Example 1.3 Multi-armed Bandit Problem

Suppose there is a fixed and finite set of n actions, a.k.a. arms, denoted \mathcal{A} . Learning proceeds in rounds, indexed by $t = 1, \dots, T$. The number of rounds T , a.k.a. the time horizon, is fixed and known in advance. In each round, the agent picks an arm a_t and observes the reward $R_t(a_t)$ for the chosen arm only. For avoidance of doubt, the agent does not observe rewards for other actions that could have been selected. If the goal is to maximize total reward over all rounds, how should the agent choose an arm?

Suppose the rewards R_t are independent and identical random variables with distribution $\nu \in [0, 1]^n$ and mean μ . The best action is then the distribution with the maximum mean μ^* .

The difference between the player’s accumulated reward and the maximum the player (a.k.a. the “cumulative regret”) could have obtained had she known all the parameters is

$$\bar{R}_T = T\mu^* - \mathbb{E} \sum_{t \in [T]} R_t.$$

Intuitively, an agent should pick arms that performed well in the past, yet the agent needs to ensure that no good option has been missed.

The theoretical origins of reinforcement learning are in stochastic dynamic programming. In this setting, an agent must make a sequence of decisions under uncertainty about the reward. If we can characterize this uncertainty with probability distributions, then the problem is typically much easier to solve. We shall assume that the reader has some familiarity with dynamic programming—the extension to stochastic dynamic programming is a relatively minor conceptual development. Note that Chap. 9 will review pertinent aspects of dynamic programming, including

Bellman optimality. The following optimal payoff example will likely just serve as a simple review exercise in dynamic programming, albeit with uncertainty introduced into the problem. As we follow the mechanics of solving the problem, the example exposes the inherent difficulty of relaxing our assumptions about the distribution of the uncertainty.

Example 1.4 Uncertain Payoffs

A strategy seeks to allocate \$600 across 3 markets and is equally profitable once the position is held, returning 1% of the size of the position over a short trading horizon $[t, t + 1]$. However, the markets vary in liquidity and there is a lower probability that the larger orders will be filled over the horizon. The amount allocated to each market must be either $K = \{100, 200, 300\}$.

Strategy	Allocation	Fill probability	Strategy	Allocation	Return
M_1	100	0.8	M_1	100	0.8
	200	0.7		200	1.4
	300	0.6		300	1.8
M_2	100	0.75	M_2	100	0.75
	200	0.7		200	1.4
	300	0.65		300	1.95
M_3	100	0.75	M_3	100	0.75
	200	0.75		200	1.5
	300	0.6		300	1.8

The optimal allocation problem under uncertainty is a stochastic dynamic programming problem. We can define value functions $v_i(x)$ for total allocation amount x for each stage of the problem, corresponding to the markets. We then find the optimal allocation using the backward recursive formulae:

$$\begin{aligned} v_3(x) &= R_3(x), \forall x \in K, \\ v_2(x) &= \max_{k \in K} \{R_2(k) + v_3(x - k)\}, \forall x \in K + 200, \\ v_1(x) &= \max_{k \in K} \{R_1(k) + v_2(x - k)\}, x = 600, \end{aligned}$$

The left-hand side of the table below tabulates the values of $R_2 + v_3$ corresponding to the second stage of the backward induction for each pair (M_2, M_3) .

(continued)

Example 1.4 (continued)

$R_2 + v_3$	M_2							
M_3	100	200	300	M_1	(M_2^*, M_3^*)	v_2	R_1	$R_1 + v_2$
100	1.5	2.15	2.7	100	(300, 200)	3.45	0.8	4.25
200	2.25	2.9	3.45	200	(200, 200)	2.9	1.4	4.3
300	2.55	3.2	3.75	300	(100, 200)	2.25	1.8	4.05

The right-hand side of the above table tabulates the values of $R_1 + v_2$ corresponding to the third and final stage of the backward induction for each tuple (M_1, M_2^*, M_3^*) .

In the above example, we can see that the allocation {200, 200, 200} maximizes the reward to give $v_1(600) = 4.3$. While this exercise is a straightforward application of a Bellman optimality recurrence relation, it provides a glimpse of the types of stochastic dynamic programming problems that can be solved with reinforcement learning. In particular, if the fill probabilities are unknown but must be learned over time by observing the outcome over each period $[t_i, t_{i+1})$, then the problem above cannot be solved by just using backward recursion. Instead we will move to the framework of reinforcement learning and attempt to learn the best actions given the data. Clearly, in practice, the example is much too simple to be representative of real-world problems in finance—the profits will be unknown and the state space is significantly larger, compounding the need for reinforcement learning. However, it is often very useful to benchmark reinforcement learning on simple stochastic dynamic programming problems with closed-form solutions.

In the previous example, we assumed that the problem was static—the variables in the problem did not change over time. This is the so-called static allocation problem and is somewhat idealized. Our next example will provide a glimpse of the types of problems that typically arise in optimal portfolio investment where random variables are dynamic. The example is also seated in more classical finance theory, that of a “Markowitz portfolio” in which the investor seeks to maximize a risk-adjusted long-term return and the wealth process is self-financing.⁶

Example 1.5 Optimal Investment in an Index Portfolio

Let S_t be a time- t price of a risky asset such as a sector exchange-traded fund (ETF). We assume that our setting is discrete time, and we denote different time steps by integer valued-indices $t = 0, \dots, T$, so there are $T + 1$ values on our discrete-time grid. The discrete-time random evolution of the risky asset S_t is

(continued)

⁶A wealth process is self-financing if, at each time step, any purchase of an additional quantity of the risky asset is funded from the bank account. Vice versa, any proceeds from a sale of some quantity of the asset go to the bank account.

Example 1.5 (continued)

$$S_{t+1} = S_t (1 + \phi_t), \quad (1.12)$$

where ϕ_t is a random variable whose probability distribution may depend on the current asset value S_t . To ensure non-negativity of prices, we assume that ϕ_t is bounded from below $\phi_t \geq -1$.

Consider a wealth process W_t of an investor who starts with an initial wealth $W_0 = 1$ at time $t = 0$ and, at each period $t = 0, \dots, T - 1$ allocates a fraction $u_t = u_t(S_t)$ of the total portfolio value to the risky asset, and the remaining fraction $1 - u_t$ is invested in a risk-free bank account that pays a risk-free interest rate $r_f = 0$. We will refer to a set of decision variables for all time steps as a *policy* $\pi := \{u_t\}_{t=0}^{T-1}$. The wealth process is self-financing and so the wealth at time $t + 1$ is given by

$$W_{t+1} = (1 - u_t) W_t + u_t W_t (1 + \phi_t). \quad (1.13)$$

This produces the one-step return

$$r_t = \frac{W_{t+1} - W_t}{W_t} = u_t \phi_t. \quad (1.14)$$

Note this is a random function of the asset price S_t . We define one-step rewards R_t for $t = 0, \dots, T - 1$ as risk-adjusted returns

$$R_t = r_t - \lambda \text{Var}[r_t | S_t] = u_t \phi_t - \lambda u_t^2 \text{Var}[\phi_t | S_t], \quad (1.15)$$

where λ is a risk-aversion parameter.^a We now consider the problem of maximization of the following concave function of the control variable u_t :

$$V^\pi(s) = \max_{u_t} \mathbb{E} \left[\sum_{t=0}^T R_t \middle| S_t = s \right] = \max_{u_t} \mathbb{E} \left[\sum_{t=0}^T u_t \phi_t - \lambda u_t^2 \text{Var}[\phi_t | S_t] \middle| S_t = s \right]. \quad (1.16)$$

Equation 1.16 defines an optimal investment problem for $T - 1$ steps faced by an investor whose objective is to optimize risk-adjusted returns over each period. This optimization problem is equivalent to maximizing the long-run

(continued)

^aNote, for avoidance of doubt, that the risk-aversion parameter must be scaled by a factor of $\frac{1}{2}$ to ensure consistency with the finance literature.

Example 1.5 (continued)

returns over the period $[0, T]$. For each $t = T - 1, T - 2, \dots, 0$, the optimality condition for action u_t is now obtained by maximization of $V^\pi(s)$ with respect to u_t :

$$u_t^* = \frac{\mathbb{E}[\phi_t | S_t]}{2\lambda \text{Var}[\phi_t | S_t]}, \quad (1.17)$$

where we allow for short selling in the ETF (i.e., $u_t < 0$) and borrowing of cash $u_t > 1$.

This is an example of a stochastic optimal control problem for a portfolio that maximizes its cumulative risk-adjusted return by repeatedly rebalancing between cash and a risky asset. Such problems can be solved using means of dynamic programming or reinforcement learning. In our problem, the dynamic programming solution is given by an analytical expression (1.17). Chapter 9 will present more complex settings including reinforcement learning approaches to optimal control problems, as well as demonstrate how expressions like the optimal action of Eq. 1.17 can be computed in practice.

? Multiple Choice Question 4

Select all the following correct statements:

1. The name “Markov processes” first historically appeared as a result of a misspelled name “Mark-Off processes” that was previously used for random processes that describe learning in certain types of video games, but has become a standard terminology since then.
2. The goal of (risk-neutral) reinforcement learning is to maximize the expected total reward by choosing an optimal policy.
3. The goal of (risk-neutral) reinforcement learning is to neutralize risk, i.e. make the variance of the total reward equal zero.
4. The goal of risk-sensitive reinforcement learning is to teach a RL agent to pick action policies that are most prone to risk of failure. Risk-sensitive RL is used, e.g. by venture capitalists and other sponsors of RL research, as a tool to assess the feasibility of new RL projects.

5 Examples of Supervised Machine Learning in Practice

The practice of machine learning in finance has grown somewhat commensurately with both theoretical and computational developments in machine learning. Early adopters have been the quantitative hedge funds, including Bridgewater Associates,

Renaissance Technologies, WorldQuant, D.E. Shaw, and Two Sigma who have embraced novel machine learning techniques although there are mixed degrees of adoption and a healthy skepticism exists that machine learning is a panacea for quantitative trading. In 2015, Bridgewater Associates announced a new artificial intelligence unit, having hired people from IBM Watson with expertise in deep learning. Anthony Ledford, chief scientist at MAN AHL: “It’s at an early stage. We have set aside a pot of money for test trading. With deep learning, if all goes well, it will go into test trading, as other machine learning approaches have.” Winton Capital Management’s CEO David Harding: “People started saying, ‘There’s an amazing new computing technique that’s going to blow away everything that’s gone before.’ There was also a fashion for genetic algorithms. Well, I can tell you none of those companies exist today—not a sausage of them.”

Some qualifications are needed to more accurately assess the extent of adoption. For instance, there is a false line of reasoning that ordinary least squares regression and logistic regression, as well as Bayesian methods, are machine learning techniques. Only if the modeling approach is algorithmic, without positing a data generation process, can the approach be correctly categorized as machine learning. So regularized regression without use of parametric assumptions on the error distribution is an example of machine learning. Unregularized regression with, say, Gaussian error is not a machine learning technique. The functional form of the input–output map is the same in both cases, which is why we emphasize that the functional form of the map is not a sufficient condition for distinguishing ML from statistical methods.

With that caveat, we shall view some examples that not only illustrate some of the important practical applications of machine learning prediction in algorithmic trading, high-frequency market making, and mortgage modeling but also provide a brief introduction to applications that will be covered in more detail in later chapters.

5.1 Algorithmic Trading

Algorithmic trading is a natural playground for machine learning. The idea behind algorithmic trading is that trading decisions should be based on data, not intuition. Therefore, it should be viable to automate this decision-making process using an algorithm, either specified or learned. The advantages of algorithmic trading include complex market pattern recognition, reduced human produced error, ability to test on historic data, etc. In recent times, as more and more information is being digitized, the feasibility and capacity of algorithmic trading has been expanding drastically. The number of hedge funds, for example, that apply machine learning for algorithmic trading is steadily increasing.

Here we provide a simple example of how machine learning techniques can be used to improve traditional algorithmic trading methods, but also provide new trading strategy suggestions. The example here is not intended to be the “best” approach, but rather indicative of more out-of-the-box strategies that machine learning facilitates, with the emphasis on minimizing out-of-sample error by pattern matching through efficient compression across high-dimensional datasets.

Momentum strategies are one of the most well-known algo-trading strategies; In general, strategies that predict prices from historic price data are categorized as momentum strategies. Traditionally momentum strategies are based on certain regression-based econometric models, such as ARIMA or VAR (see Chap. 6). A drawback of these models is that they impose strong linearity which is not consistently plausible for time series of prices. Another caveat is that these models are parametric and thus have strong bias which often causes underfitting. Many machine learning algorithms are both non-linear and semi/non-parametric, and therefore prove complementary to existing econometric models.

In this example we build a simple momentum portfolio strategy with a feedforward neural network. We focus on the S&P 500 stock universe, and assume we have daily close prices for all stocks over a ten-year period.⁷

Problem Formulation

The most complex practical aspect of machine learning is how to choose the input (“features”) and output. The type of desired output will determine whether a regressor or classifier is needed, but the general rule is that it must be actionable (i.e., tradable). Suppose our goal is to invest in an equally weighted, long only, stock portfolio only if it beats the S&P 500 index benchmark (which is a reasonable objective for a portfolio manager). We can therefore label the portfolio at every observation t based on the mean directional excess return of the portfolio:

$$G_t = \begin{cases} 1 & \frac{1}{N} \sum_i r_{t+h,t}^i - \tilde{r}_{t+h,t} \geq \epsilon, \\ 0 & \frac{1}{N} \sum_i r_{t+h,t}^i - \tilde{r}_{t+h,t} < 0, \end{cases} \quad (1.18)$$

where $r_{t+h,t}^i$ is the return of stock i between times t and $t + h$, $\tilde{r}_{t+h,t}$ is the return of the S&P 500 index in the same period, and ϵ is some target next period excess portfolio return. Without loss of generality, we could invest in the universe ($N = 500$), although this is likely to have adverse practical implications such as excessive transaction costs. We could easily just have restricted the number of stocks to a subset, such as the top decile of performing stocks in the last period. Framed this way, the machine learner is thus informing us when our stock selection strategy will outperform the market. It is largely agnostic to how the stocks are selected, provided the procedure is systematic and based solely on the historic data provided to the classifier. It is further worth noting that the map between the decision to hold the customized portfolio has a non-linear relationship with the past returns of the universe.

To make the problem more concrete, let us set $h = 5$ days. The algorithmic strategy here is therefore automating the decision to invest in the customized

⁷The question of how much data is needed to train a neural network is a central one, with the immediate concern being insufficient data to avoid over-fitting. The amount of data needed is complex to assess; however, it is partly dependent on the number of edges in the network and can be assessed through bias-variance analysis, as described in Chap. 4.

Table 1.3 Training samples for a classification problem

Date	X_1	X_2	...	X_{500}	G
2007-01-03	0.051	-0.035		0.072	0
2017-01-04	-0.092	0.125		-0.032	0
2017-01-05	0.021	0.063		-0.058	1
...					
2017-12-29	0.093	-0.023		0.045	1
2017-12-30	0.020	0.019		0.022	1
2017-12-31	-0.109	0.025		-0.092	1

portfolio or the S&P 500 index every week based on the previous 5-day realized returns of all stocks. To apply machine learning to this decision, the problem translates into finding the weights in the network between past returns and the decision to invest in the equally weighted portfolio. For avoidance of doubt, we emphasize that the interpretation of the optimal weights differs substantially from Markowitz's mean-variance portfolios, which simply finds the portfolio weights to optimize expected returns for a given risk tolerance. Here, we either invest equal amounts in all stocks of the portfolio or invest the same amount in the S&P 500 index and the weights in the network signify the relevance of past stock returns in the expected excess portfolio return outperforming the market.

Data

Feature engineering is always important in building models and requires careful consideration. Since the original price data does not meet several machine learning requirements, such as stationarity and i.i.d. distributional properties, one needs to engineer input features to prevent potential "garbage-in-garbage-out" phenomena. In this example, we take a simple approach by using only the 5-day realized returns of all S&P 500 stocks.⁸ Returns are scale-free and no further standardization is needed. So for each time t , the input features are

$$\mathbf{X}_t = \left[r_{t,t-5}^1, \dots, r_{t,t-5}^{500} \right]. \quad (1.19)$$

Now we can aggregate the features and labels into a panel indexed by date. Each column is an entry in Eq. 1.19, except for the last column which is the assigned label from Eq. 1.18, based on the realized excess stock returns of the portfolio. An example of the labeled input data (X, G) is shown in Table 1.3.

The process by which we train the classifier and evaluate its performance will be described in Chap. 4, but this example illustrates how algo-trading strategies can be crafted around supervised machine learning. Our model problem could be tailored

⁸Note that the composition of the S&P 500 changes over time and so we should interpret a feature as a fixed symbol.

for specific risk-reward and performance reporting metrics such as, for example, Sharpe or information ratios meeting or exceeding a threshold.

ϵ is typically chosen to be a small value so that the labels are not too imbalanced. As the value ϵ is increased, the problem becomes an “outlier prediction problem”—a highly imbalanced classification problem which requires more advanced sampling and interpolation techniques beyond an off-the-shelf classifier.

In the next example, we shall turn to another important aspect of machine learning in algorithmic trading, namely execution. How the trades are placed is a significant aspect of algorithmic trading strategy performance, not only to minimize price impact of market taking strategies but also for market making. Here we shall look to transactional data to perfect the execution, an engineering challenge by itself just to process market feeds of tick-by-tick exchange transactions. The example considers a market making application but could be adapted for price impact and other execution considerations in algorithmic trading by moving to a reinforcement learning framework.

A common mistake is to assume that building a predictive model will result in a profitable trading strategy. Clearly, the consideration given to reliably evaluating machine learning in the context of trading strategy performance is a critical component of its assessment.

5.2 High-Frequency Trade Execution

Modern financial exchanges facilitate the electronic trading of instruments through an instantaneous double auction. At each point in time, the market demand and the supply can be represented by an electronic limit order book, a cross-section of orders to execute at various price levels away from the market price as illustrated in Table 1.4.

Electronic market makers will quote on both sides of the market in an attempt to capture the bid–ask spread. Sometimes a large market order, or a succession of smaller markets orders, will consume an entire price level. This is why the market price fluctuates in liquid markets—an effect often referred to by practitioners as a “price-flip.” A market maker can take a loss if only one side of the order is filled as a result of an adverse price movement.

Figure 1.7 (left) illustrates a typical mechanism resulting in an adverse price movement. A snapshot of the limit order book at time t , before the arrival of a market order, and after at time $t+1$ is shown in the left and right panels, respectively. The resting orders placed by the market marker are denoted with the “+” symbol—red denotes a bid and blue denotes an ask quote. A buy market order subsequently arrives and matches the entire resting quantity of best ask quotes. Then at event time $t+1$ the limit order book is updated—the market maker’s ask has been filled (blue minus symbol) and the bid now rests away from the inside market. The market marker may systematically be forced to cancel the bid and buy back at a higher price, thus taking a loss.

Table 1.4 This table shows a snapshot of the limit order book of S&P 500 e-mini futures (ES). The top half (“sell-side”) shows the ask volumes and prices and the lower half (“buy side”) shows the bid volumes and prices. The quote levels are ranked by the most competitive at the center (the “inside market”), outward to the least competitive prices at the top and bottom of the limit order book. Note that only five bid or ask levels are shown in this example, but the actual book is much deeper

Bid	Price	Ask
	2170.25	1284
	2170.00	1642
	2169.75	1401
	2169.50	1266
	2169.25	290
477	2169.00	
1038	2168.75	
950	2168.50	
1349	2168.25	
1559	2168.00	

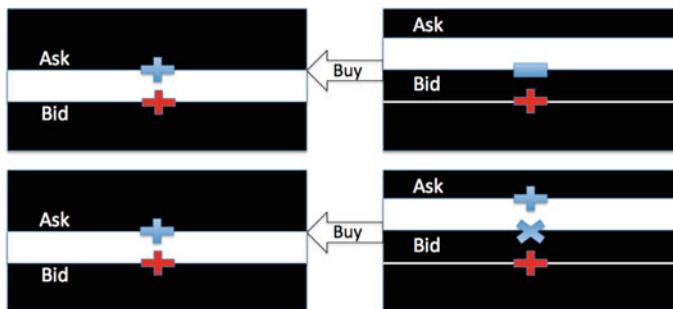


Fig. 1.7 (Top) A snapshot of the limit order book is taken at time t . Limit orders placed by the market marker are denoted with the “+” symbol—red denotes a bid and blue denotes an ask. A buy market order subsequently arrives and matches the entire resting quantity of best ask quotes. Then at event time $t + 1$ the limit order book is updated. The market maker’s ask has been filled (blue minus symbol) and the bid rests away from the inside market. (Bottom) A pre-emptive strategy for avoiding adverse price selection is illustrated. The ask is requoted at a higher ask price. In this case, the bid is not replaced and the market maker may capture a tick more than the spread if both orders are filled

Machine learning can be used to predict these price movements (Kearns and Nevmyvaka 2013; Kercheval and Zhang 2015; Sirignano 2016; Dixon et al. 2018; Dixon 2018b,a) and thus to potentially avoid adverse selection. Following Cont and de Larrard (2013) we can treat queue sizes at each price level as input variables. We can additionally include properties of market orders, albeit in a form which our machines deem most relevant to predicting the direction of price movements (a.k.a. feature engineering). In contrast to stochastic modeling, we do not impose conditional distributional assumptions on the independent variables (a.k.a. features) nor assume that price movements are Markovian. Chapter 8 presents a RNN for

mid-price prediction from the limit order book history which is the starting point for the more in-depth study of Dixon (2018b) which includes market orders and demonstrates the superiority of RNNs compared to other time series methods such as Kalman filters.

We reiterate that the ability to accurately predict does not imply profitability of the strategy. Complex issues concerning queue position, exchange matching rules, latency, position constraints, and price impact are central considerations for practitioners. The design of profitable strategies goes beyond the scope of this book but the reader is referred to de Prado (2018) for the pitfalls of backtesting and designing algorithms for trading. Dixon (2018a) presents a framework for evaluating the performance of supervised machine learning algorithms which accounts for latency, position constraints, and queue position. However, supervised learning is ultimately not the best machine learning approach as it cannot capture the effect of market impact and is too inflexible to incorporate more complex strategies. Chapter 9 presents examples of reinforcement learning which demonstrate how to capture market impact and also how to flexibly formulate market making strategies.

5.3 Mortgage Modeling

Beyond the data rich environment of algorithmic trading, does machine learning have a place in finance? One perspective is that there simply is not sufficient data for some “low-frequency” application areas in finance, especially where traditional models have failed catastrophically. The purpose of this section is to serve as a sobering reminder that long-term forecasting goes far beyond merely selecting the best choice of machine learning algorithm and why there is no substitute for strong domain knowledge and an understanding of the limitations of data.

In the USA, a mortgage is a loan collateralized by real-estate. Mortgages are used to securitize financial instruments such as mortgage backed securities and collateralized mortgage obligations. The analysis of such securities is complex and has changed significantly over the last decade in response to the 2007–2008 financial crises (Stein 2012).

Unless otherwise specified, a mortgage will be taken to mean a “residential mortgage,” which is a loan with payments due monthly that is collateralized by a single family home. Commercial mortgages do exist, covering office towers, rental apartment buildings, and industrial facilities, but they are different enough to be considered separate classes of financial instruments. Borrowing money to buy a house is one of the most common, and largest balance, loans that an individual borrower is ever likely to commit to. Within the USA alone, mortgages comprise a staggering \$15 trillion dollars in debt. This is approximately the same balance as the total federal debt outstanding (Fig. 1.8).

Within the USA, mortgages may be repaid (typically without penalty) at will by the borrower. Usually, borrowers use this feature to refinance their loans in favorable interest rate regimes, or to liquidate the loan when selling the underlying house. This

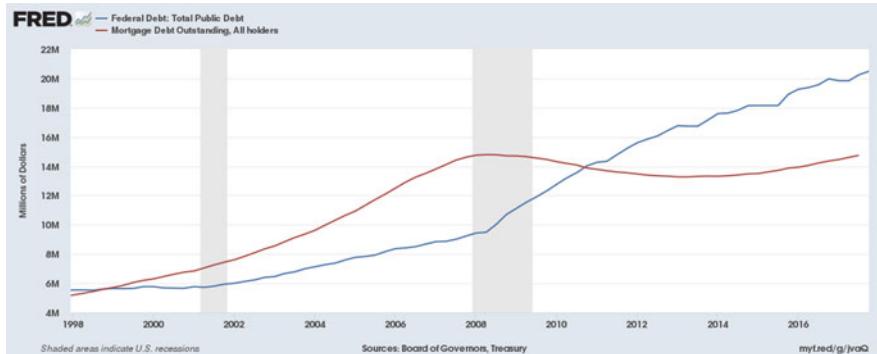


Fig. 1.8 Total mortgage debt in the USA compared to total federal debt, millions of dollars, unadjusted for inflation. Source: <https://fred.stlouisfed.org/series/MDOAH>, <https://fred.stlouisfed.org/series/GFDEBTN>

Table 1.5 At any time, the states of any US style residential mortgage is in one of the several possible states

Symbol	Name	Definition
P	Paid	All balances paid, loan is dissolved
C	Current	All payments due have been paid
3	30-days delinquent	Mortgage is delinquent by one payment
6	60-days delinquent	Delinquent by 2 payments
9	90+ delinquent	Delinquent by 3 or more payments
F	Foreclosure	Foreclosure has been initiated by the lender
R	Real-Estate-Owned (REO)	The lender has possession of the property
D	Default liquidation	Loan is involuntarily liquidated for nonpayment

has the effect of moving a great deal of financial risk off of individual borrowers, and into the financial system. It also drives a lively and well developed industry around modeling the behavior of these loans.

The mortgage model description here will generally follow the comprehensive work in Sirignano et al. (2016), with only a few minor deviations.

Any US style residential mortgage, in each month, can be in one of the several possible states listed in Table 1.5.

Consider this set of K available states to be $\mathbb{K} = \{P, C, 3, 6, 9, F, R, D\}$. Following the problem formulation in Sirignano et al. (2016), we will refer to the status of loan n at time t as $U_t^n \in \mathbb{K}$, and this will be represented as a probability vector using a standard one-hot encoding.

If $X = (X_1, \dots, X_P)$ is the input matrix of P explanatory variables, then we define a probability transition density function $g : \mathbb{R}^P \rightarrow [0, 1]^{K \times K}$ parameterized by θ so that

$$\mathbb{P}(U_{t+1}^n = i \mid U_t^n = j, X_t^n) = g_{i,j}(X_t^n \mid \theta), \forall i, j \in \mathbb{K}. \quad (1.20)$$

Note that $g(X_t^n | \theta)$ is a time in-homogeneous $K \times K$ Markov transition matrix. Also, not all transitions are even conceptually possible—there are non-commutative states. For instance, a transition from C to 6 is not possible since a borrower cannot miss two payments in a single month. Here we will write $p_{(i,j)} := g_{i,j}(X_t^n | \theta)$ for ease of notation and because of the non-commutative state transitions where $p_{(i,j)} = 0$, the Markov matrix takes the form:

$$g(X_t^n | \theta) = \begin{bmatrix} 1 & p_{(c,p)} & p_{(3,p)} & 0 & 0 & 0 & 0 & 0 \\ 0 & p_{(c,c)} & p_{(3,c)} & p_{(6,c)} & p_{(9,c)} & p_{(f,c)} & 0 & 0 \\ 0 & p_{(c,3)} & p_{(3,3)} & p_{(6,3)} & p_{(9,3)} & p_{(f,3)} & 0 & 0 \\ 0 & 0 & p_{(3,6)} & p_{(6,6)} & p_{(9,6)} & p_{(f,6)} & 0 & 0 \\ 0 & 0 & 0 & p_{(6,9)} & p_{(9,9)} & p_{(f,9)} & 0 & 0 \\ 0 & 0 & 0 & p_{(6,f)} & p_{(9,f)} & p_{(f,f)} & 0 & 0 \\ 0 & 0 & 0 & p_{(6,r)} & p_{(9,r)} & p_{(f,r)} & p_{(r,r)} & 0 \\ 0 & 0 & 0 & p_{(6,d)} & p_{(9,d)} & p_{(f,d)} & p_{(r,d)} & 1 \end{bmatrix}.$$

Our classifier $g_{i,j}(X_t^n | \theta)$ can thus be constructed so that only the probability of transition between the commutative states are outputs and we can apply softmax functions on a subset of the outputs to ensure that $\sum_{j \in \mathbb{K}} g_{i,j}(X_t^n | \theta) = 1$ and hence the transition probabilities in each row sum to one.

For the purposes of financial modeling, it is important to realize that both states P and D are loan liquidation terminal states. However, state P is considered to be voluntary loan liquidation (e.g., prepayment due to refinance), whereas state D is considered to be involuntary liquidation (e.g., liquidation via foreclosure and auction). These states are not distinguishable in the mortgage data itself, but rather the driving force behind liquidation must be inferred from the events leading up to the liquidation.

One contributor to mortgage model misprediction in the run up to the 2008 financial crisis was that some (but not all) modeling groups considered loans liquidating from deep delinquency (e.g., status 9) to be the transition $9 \rightarrow P$ if no losses were incurred. However, behaviorally, these were typically defaults due to financial hardship, and they would have had losses in a more difficult house price regime. They were really $9 \rightarrow D$ transitions that just happened to be lossless due to strong house price gains over the life of the mortgage. Considering them to be voluntary prepayments (status P) resulted in systematic over-prediction of prepayments in the aftermath of major house price drops. The matrix above therefore explicitly excludes this possibility and forces delinquent loan liquidation to be always considered involuntary.

The reverse of this problem does not typically exist. In most states it is illegal to force a borrower into liquidation until at least 2 payments have been missed. Therefore, liquidation from C or 3 is always voluntary, and hence $C \rightarrow P$ and $3 \rightarrow P$. Except in cases of fraud or severe malfeasance, it is almost never economically advantageous for a lender to force liquidation from status 6, but it is not illegal. Therefore the transition $3 \rightarrow D$ is typically a data error, but $6 \rightarrow D$ is merely very rare.

Example 1.6 Parameterized Probability Transitions

If loan n is current in time period t , then

$$\mathbb{P}(U_t^n) = (0, 1, 0, 0, 0, 0, 0, 0)^T. \quad (1.21)$$

If we have $p_{(c,p)} = 0.05$, $p_{(c,c)} = 0.9$, and $p_{(c,3)} = 0.05$, then

$$\mathbb{P}(U_{t+1}^n | X_t^n) = g(X_t^n | \theta) \cdot \mathbb{P}(U_t^n) = (0.05, 0.9, 0.05, 0, 0, 0, 0, 0)^T. \quad (1.22)$$

Common mortgage models sometimes use additional states, often ones that are (without additional information) indistinguishable from the states listed above. Table 1.6 describes a few of these.

The reason for including these is the same as the reason for breaking out states like REO, status R . It is known on theoretical grounds that some model regressors from X_t^n should not be relevant for R . For instance, since the property is now owned by the lender, and the loan itself no longer exists, the interest rate (and rate incentive) of the original loan should no longer have a bearing on the outcome. To avoid over-fitting due to highly colinear variables, these known-useless variables are then excluded from transitions models starting in status R .

This is the same reason status T is sometimes broken out, especially for logistic regressions. Without an extra status listed in this way, strong rate disincentives could drive prepayments in the model to (almost) zero, but we know that people die and divorce in all rate regimes, so at least some minimal level of premature loan liquidations must still occur based on demographic factors, not financial ones.

5.3.1 Model Stability

Unlike many other models, mortgage models are designed to accurately predict events a decade or more in the future. Generally, this requires that they be built on regressors that themselves can be accurately predicted, or at least hedged. Therefore, it is common to see regressors like FICO at origination, loan age in months, rate incentive, and loan-to-value (LTV) ratio. Often LTV would be called MTMLTV if it is marked-to-market against projected or realized housing price moves. Of these regressors, original FICO is static over the life of the loan, age is deterministic,

Table 1.6 A brief description of mortgage states

Symbol	Name	Overlaps with	Definition
T	Turnover	P	Loan prepaid due to non-financial life event
U	Curtailment	C	Borrower overpaid to reduce loan principal

Table 1.7 Loan originations by year (Freddie Mac, FRM30)

Year	Loans originated
1999	976,159
2000	733,567
2001	1,542,025
2002	1,403,515
2003	2,063,488
2004	1,133,015
2005	1,618,748
2006	1,300,559
2007	1,238,814
2008	1,237,823
2009	1,879,477
2010	1,250,484
2011	1,008,731
2012	1,249,486
2013	1,375,423
2014	942,208

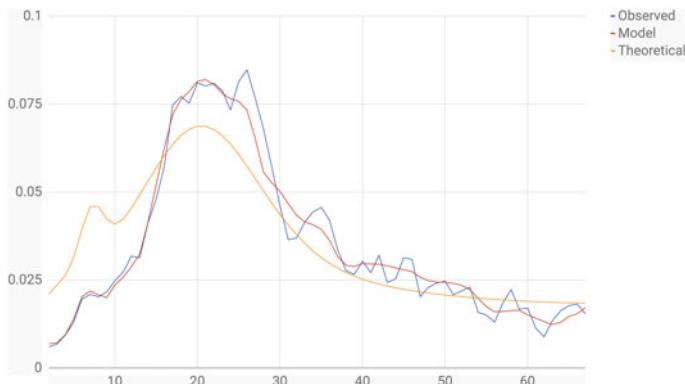


Fig. 1.9 Sample mortgage model predicting $C \rightarrow 3$ and fit on loans originated in 2001 and observed until 2006, by loan age (in months). The prepayment probabilities are shown on the y-axis

rates can be hedged, and MTMLTV is rapidly driven down by loan amortization and inflation thus eliminating the need to predict it accurately far into the future.

Consider the Freddie Mac loan level dataset of 30 year fixed rate mortgages originated through 2014. This includes each monthly observation from each loan present in the dataset. Table 1.7 shows the loan count by year for this dataset.

When a model is fit on 1 million observations from loans originated in 2001 and observed until the end of 2006, its $C \rightarrow P$ probability charted against age is shown in Fig. 1.9.

In Fig. 1.9 the curve observed is the actual prepayment probability of the observations with the given age in the test dataset, “Model” is the model prediction,

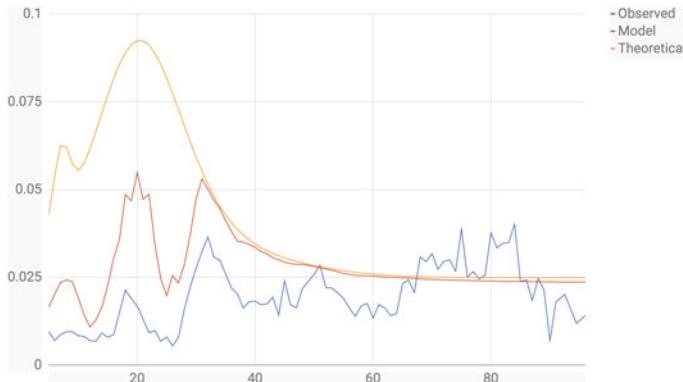


Fig. 1.10 Sample mortgage model predicting $C \rightarrow 3$ and fit on loans originated in 2006 and observed until 2015, by loan age (in months). The prepayment probabilities are shown on the y-axis

and “Theoretical” is the response to age by a theoretical loan with all other regressors from X_t^n held constant. Two observations are worth noting:

1. The marginal response to age closely matches the model predictions; and
2. The model predictions match actual behavior almost perfectly.

This is a regime where prepayment behavior is largely driven by age. When that same model is run on observations from loans originated in 2006 (the peak of housing prices before the crisis), and observed until 2015, Fig. 1.10 is produced.

Three observations are warranted from this figure:

1. The observed distribution is significantly different from Fig. 1.9;
2. The model predicted a decline of 25%, but the actual decline was approximately 56%; and
3. Prepayment probabilities are largely indifferent to age.

The regime shown here is clearly not driven by age. In order to provide even this level of accuracy, the model had to extrapolate far from any of the available data and “imagine” a regime where loan age is almost irrelevant to prepayment. This model meets with mixed success. This particular one was fit on only 8 regressors, a more complicated model might have done better, but the actual driver of this inaccuracy was a general tightening of lending standards. Moreover, there was no good data series available before the crisis to represent lending standards.

This model was reasonably accurate even though almost 15 years separated the start of the fitting data from the end of the projection period, and a lot happened in that time. Mortgage models in particular place a high premium on model stability, and the ability to provide as much accuracy as possible even though the underlying distribution may have changed dramatically from the one that generated the fitting data. Notice also that cross-validation would not help here, as we cannot draw testing data from the distribution we care about, since that distribution comes from the future.

Most importantly, this model shows that the low-dimensional projections of this (moderately) high-dimensional problem are extremely deceptive. No modeler would have chosen a shape like the model prediction from Fig. 1.9 as function of age. That prediction arises due to the interaction of several variables, interactions that are not interpretable from one-dimensional plots such as this. As we will see in subsequent chapters, such complexities in data are well suited to machine learning, but not without a cost. That cost is understanding the “bias–variance tradeoff” and understanding machine learning with sufficient rigor for its decisions to be defensible.

6 Summary

In this chapter, we have identified some of the key elements of supervised machine learning. Supervised machine learning

1. is an algorithmic approach to statistical inference which, crucially, does not depend on a data generation process;
2. estimates a parameterized map between inputs and outputs, with the functional form defined by the methodology such as a neural network or a random forest;
3. automates model selection, using regularization and ensemble averaging techniques to iterate through possible models and arrive at a model with the best out-of-sample performance; and
4. is often well suited to large sample sizes of high-dimensional non-linear covariates.

The emphasis on out-of-sample performance, automated model selection, and absence of a pre-determined parametric data generation process is really the key to machine learning being a more robust approach than many parametric, financial econometrics techniques in use today. The key to adoption of machine learning in finance is the ability to run machine learners alongside their parametric counterparts, observing over time the differences and limitations of parametric modeling based on in-sample fitting metrics. Statistical tests must be used to characterize the data and guide the choice of algorithm, such as, for example, tests for stationarity. See Dixon and Halperin (2019) for a checklist and brief but rounded discussion of some of the challenges in adopting machine learning in the finance industry.

Capacity to readily exploit a wide form of data is their other advantage, but only if that data is sufficiently high quality and adds a new source of information. We close this chapter with a reminder of the failings of forecasting models during the financial crisis of 2008 and emphasize the importance of avoiding siloed data extraction. The application of machine learning requires strong scientific reasoning skills and is not a panacea for commoditized and automated decision-making.

7 Exercises

Exercise 1.1**: Market Game

Suppose that two players enter into a market game. The rules of the game are as follows: Player 1 is the market maker, and Player 2 is the market taker. In each round, Player 1 is provided with information \mathbf{x} , and must choose and declare a value $\alpha \in (0, 1)$ that determines how much it will pay out if a binary event G occurs in the round. $G \sim \text{Bernoulli}(p)$, where $p = g(\mathbf{x}|\boldsymbol{\theta})$ for some unknown parameter $\boldsymbol{\theta}$.

Player 2 then enters the game with a \$1 payment and chooses one of the following payoffs:

$$V_1(G, p) = \begin{cases} \frac{1}{\alpha} & \text{with probability } p \\ 0 & \text{with probability } (1 - p) \end{cases}$$

or

$$V_2(G, p) = \begin{cases} 0 & \text{with probability } p \\ \frac{1}{(1-\alpha)} & \text{with probability } (1 - p) \end{cases}$$

- Given that α is known to Player 2, state the strategy⁹ that will give Player 2 an expected payoff, over multiple games, of \$1 without knowing p .
- Suppose now that p is known to both players. In a given round, what is the optimal choice of α for Player 1?
- Suppose Player 2 knows with complete certainty, that G will be 1 for a particular round, what will be the payoff for Player 2?
- Suppose Player 2 has complete knowledge in rounds $\{1, \dots, i\}$ and can reinvest payoffs from earlier rounds into later rounds. Further suppose without loss of generality that $G = 1$ for each of these rounds. What will be the payoff for Player 2 after i rounds? You may assume that the each game can be played with fractional dollar costs, so that, for example, if Player 2 pays Player 1 \$1.5 to enter the game, then the payoff will be $1.5V_1$.

Exercise 1.2**: Model Comparison

Recall Example 1.2. Suppose additional information was added such that it is no longer possible to predict the outcome with 100% probability. Consider Table 1.8 as the results of some experiment.

Now if we are presented with $\mathbf{x} = (1, 0)$, the result could be B or C . Consider three different models applied to this value of \mathbf{x} which encode the value A, B, or C.

⁹The strategy refers the choice of weight if Player 2 is to choose a payoff $V = wV_1 + (1 - w)V_2$, i.e. a weighted combination of payoffs V_1 and V_2 .

Table 1.8 Sample model data

G	x
A	(0, 1)
B	(1, 1)
B	(1, 0)
C	(1, 0)
C	(0, 0)

$$f((1, 0)) = (0, 1, 0), \text{ Predicts B with 100\% certainty.} \quad (1.23)$$

$$g((1, 0)) = (0, 0, 1), \text{ Predicts C with 100\% certainty.} \quad (1.24)$$

$$h((1, 0)) = (0, 0.5, 0.5), \text{ Predicts B or C with 50\% certainty.} \quad (1.25)$$

1. Show that each model has the same total absolute error, over the samples where $x = (1, 0)$.
2. Show that all three models assign the same average probability to the values from Table 1.8 when $x = (1, 0)$.
3. Suppose that the market game in Exercise 1 is now played with models f or g . B or C each triggers two separate payoffs, V_1 and V_2 , respectively. Show that the losses to Player 1 are unbounded when $x = (1, 0)$ and $\alpha = 1 - p$.
4. Show also that if the market game in Exercise 1 is now played with model h , the losses to Player 1 are bounded.

Exercise 1.3**: Model Comparison

Example 1.1 and the associated discussion alluded to the notion that some types of models are more common than others. This exercise will explore that concept briefly.

Recall Table 1.1 from Example 1.1:

G	x
A	(0, 1)
B	(1, 1)
C	(1, 0)
C	(0, 0)

For this exercise, consider two models “similar” if they produce the same projections for G when applied to the values of x from Table 1.1 with probability strictly greater than 0.95.

In the following subsections, the goal will be to produce sets of mutually dissimilar models that all produce Table 1.1 with a given likelihood.

1. How many similar models produce Table 1.1 with likelihood 1.0?
2. Produce at least 4 dissimilar models that produce Table 1.1 with likelihood 0.9.
3. How many dissimilar models can produce Table 1.1 with likelihood exactly 0.95?

Exercise 1.4*: Likelihood Estimation

When the data is i.i.d., the negative of log-likelihood function (the “error function”) for a binary classifier is the *cross-entropy*

$$E(\boldsymbol{\theta}) = - \sum_{i=1}^n G_i \ln(g_1(\mathbf{x}_i | \boldsymbol{\theta})) + (1 - G_i) \ln(g_0(\mathbf{x}_i | \boldsymbol{\theta})).$$

Suppose now that there is a probability π_i that the class label on a training data point \mathbf{x}_i has been correctly set. Write down the error function corresponding to the negative log-likelihood. Verify that the error function in the above equation is obtained when $\pi_i = 1$. Note that this error function renders the model robust to incorrectly labeled data, in contrast to the usual least squares error function.

Exercise 1.5: Optimal Action**

Derive Eq. 1.17 by setting the derivative of Eq. 1.16 with respect to the time- t action u_t to zero. Note that Eq. 1.17 gives a non-parametric expression for the optimal action u_t in terms of a ratio of two conditional expectations. To be useful in practice, the approach might need some further modification as you will use in the next exercise.

Exercise 1.6*: Basis Functions**

Instead of non-parametric specifications of an optimal action in Eq. 1.17, we can develop a *parametric* model of optimal action. To this end, assume we have a set of basic functions $\psi_k(S)$ with $k = 1, \dots, K$. Here K is the total number of basis functions—the same as the dimension of your model space.

We now define the optimal action $u_t = u_t(S_t)$ in terms of coefficients θ_k of expansion over basis functions Ψ_k (for example, we could use spline basis functions, Fourier bases, etc.) :

$$u_t = u_t(S_t) = \sum_{k=1}^K \theta_k(t) \Psi_k(S_t).$$

Compute the optimal coefficients $\theta_k(t)$ by substituting the above equation for u_t into Eq. 1.16 and maximizing it with respect to a set of weights $\theta_k(t)$ for a t -th time step.

Appendix

Answers to Multiple Choice Questions

Question 1

Answer: 1, 2.

Answer 3 is incorrect. While it is true that unsupervised learning does not require a human supervisor to train the model, it is false to presume that the approach is superior.

Answer 4 is incorrect. Reinforcement learning cannot be viewed as a generalization of supervised learning to Markov Decision Processes. The reason is that reinforcement learning uses rewards to reinforce decisions, rather than labels to define the correct decision. For this reason, reinforcement learning uses a weaker form of supervision.

Question 2

Answer: 1,2,3.

Answer 4 is incorrect. Two separate binary models $\{g_i^{(1)}(X|\theta)\}_{i=0}^1$ and $\{g_i^{(2)}(X|\theta)\}_{i=0}^1$ will, in general, not produce the same output as a single, multi-class, model $\{g_i(X|\theta)\}_{i=0}^3$. Consider, as a counter example, the logistic models $g_0^{(1)} = g_0(X|\theta_1) = \frac{\exp\{-X^T\theta_1\}}{1+\exp\{-X^T\theta_1\}}$ and $g_0^{(2)} = g_0(X|\theta_2) = \frac{\exp\{-X^T\theta_2\}}{1+\exp\{-X^T\theta_2\}}$, compared with the multi-class model

$$g_i(X|\theta') = \text{softmax}(\exp\{X^T\theta'\}) = \frac{\exp\{(X^T\theta')_i\}}{\sum_{k=0}^K \exp\{(X^T\theta')_k\}}. \quad (1.26)$$

If we set $\theta_1 = \theta'_0 - \theta'_1$ and $\theta'_2 = \theta'_3 = 0$, then the multi-class model is equivalent to Model 1. Similarly if we set $\theta_2 = \theta'_2 - \theta'_3$ and $\theta'_0 = \theta'_1 = 0$, then the multi-class model is equivalent to Model 2. However, we cannot simultaneously match the outputs of Model 1 and Model 2 with the multi-class model.

Question 3

Answer: 1,2,3.

Answer 4 is incorrect. The layers in a deep recurrent network provide more expressibility between each lagged input and the hidden state variable, but are unrelated to the amount of memory in the network. The hidden layers in any multilayered perceptron are not the hidden state variables in our time series model. It is the degree of unfolding, i.e. number of hidden state vectors which determines the amount of memory in any recurrent network.

Question 4

Answer: 2.

References

- Akaike, H. (1973). *Information theory and an extension of the maximum likelihood principle* (pp. 267–281).
- Akcora, C. G., Dixon, M. F., Gel, Y. R., & Kantarcioglu, M. (2018). Bitcoin risk modeling with blockchain graphs. *Economics Letters*, 173(C), 138–142.
- Arnold, V. I. (1957). *On functions of three variables* (Vol. 114, pp. 679–681).
- Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31, 307–327.
- Box, G. E. P., & Jenkins, G. M. (1976). *Time series analysis, forecasting, and control*. San Francisco: Holden-Day.

- Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (1994). *Time series analysis, forecasting, and control* (third ed.). Englewood Cliffs, NJ: Prentice-Hall.
- Breiman, L. (2001). Statistical modeling: the two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3), 199–231.
- Cont, R., & de Larrard, A. (2013). Price dynamics in a Markovian limit order market. *SIAM Journal on Financial Mathematics*, 4(1), 1–25.
- de Prado, M. (2018). *Advances in financial machine learning*. Wiley.
- de Prado, M. L. (2019). Beyond econometrics: A roadmap towards financial machine learning. SSRN. Available at SSRN: <https://ssrn.com/abstract=3365282> or <http://dx.doi.org/10.2139/ssrn.3365282>.
- DeepMind (2016). DeepMind AI reduces Google data centre cooling bill by 40%. <https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/>.
- DeepMind (2017). The story of AlphaGo so far. <https://deepmind.com/research/alphago/>.
- Dhar, V. (2013, December). Data science and prediction. *Commun. ACM*, 56(12), 64–73.
- Dixon, M. (2018a). A high frequency trade execution model for supervised learning. *High Frequency*, 1(1), 32–52.
- Dixon, M. (2018b). Sequence classification of the limit order book using recurrent neural networks. *Journal of Computational Science*, 24, 277–286.
- Dixon, M., & Halperin, I. (2019). *The four horsemen of machine learning in finance*.
- Dixon, M., Polson, N., & Sokolov, V. (2018). Deep learning for spatio-temporal modeling: Dynamic traffic flows and high frequency trading. *ASMB*.
- Dixon, M. F., & Polson, N. G. (2019, Mar). Deep fundamental factor models. *arXiv e-prints*, arXiv:1903.07677.
- Dyhrberg, A. (2016). Bitcoin, gold and the dollar – a GARCH volatility analysis. *Finance Research Letters*.
- Elman, J. L. (1991, Sep). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2), 195–225.
- Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., et al. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639), 115–118.
- Flood, M., Jagadish, H. V., & Raschid, L. (2016). Big data challenges and opportunities in financial stability monitoring. *Financial Stability Review*, (20), 129–142.
- Gomber, P., Koch, J.-A., & Siering, M. (2017). Digital finance and fintech: current research and future research directions. *Journal of Business Economics*, 7(5), 537–580.
- Gottlieb, O., Salisbury, C., Shek, H., & Vaidyanathan, V. (2006). Detecting corporate fraud: An application of machine learning. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.142.7470>.
- Graves, A. (2012). *Supervised sequence labelling with recurrent neural networks*. Studies in Computational intelligence. Heidelberg, New York: Springer.
- Gu, S., Kelly, B. T., & Xiu, D. (2018). *Empirical asset pricing via machine learning*. Chicago Booth Research Paper 18–04.
- Harvey, C. R., Liu, Y., & Zhu, H. (2016). ... and the cross-section of expected returns. *The Review of Financial Studies*, 29(1), 5–68.
- Hornik, K., Stinchcombe, M., & White, H. (1989, July). Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5), 359–366.
- Kearns, M., & Nevinmyvaka, Y. (2013). Machine learning for market microstructure and high frequency trading. *High Frequency Trading - New Realities for Traders*.
- Kercheval, A., & Zhang, Y. (2015). Modeling high-frequency limit order book dynamics with support vector machines. *Journal of Quantitative Finance*, 15(8), 1315–1329.
- Kolmogorov, A. N. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk SSSR*, 114, 953–956.
- Kubota, T. (2017, January). Artificial intelligence used to identify skin cancer.

- Kullback, S., & Leibler, R. A. (1951, 03). On information and sufficiency. *Ann. Math. Statist.*, 22(1), 79–86.
- McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (1955, August). A proposal for the Dartmouth summer research project on artificial intelligence. <http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html>.
- Philipp, G., & Carbonell, J. G. (2017, Dec). Nonparametric neural networks. *arXiv e-prints*, arXiv:1712.05440.
- Philippon, T. (2016). *The fintech opportunity*. CEPR Discussion Papers 11409, C.E.P.R. Discussion Papers.
- Pinar Saygin, A., Cicekli, I., & Akman, V. (2000, November). Turing test: 50 years later. *Minds Mach.*, 10(4), 463–518.
- Poggio, T. (2016). Deep learning: mathematics and neuroscience. *A Sponsored Supplement to Science Brain-Inspired intelligent robotics: The intersection of robotics and neuroscience*, 9–12.
- Shannon, C. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27.
- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.
- Sirignano, J., Sadhwani, A., & Giesecke, K. (2016, July). Deep learning for mortgage risk. *ArXiv e-prints*.
- Sirignano, J. A. (2016). Deep learning for limit order books. *arXiv preprint arXiv:1601.01987*.
- Sovbetov, Y. (2018). Factors influencing cryptocurrency prices: Evidence from Bitcoin, Ethereum, Dash, Litecoin, and Monero. *Journal of Economics and Financial Analysis*, 2(2), 1–27.
- Stein, H. (2012). Counterparty risk, CVA, and Basel III.
- Turing, A. M. (1995). *Computers & thought*. Chapter Computing Machinery and Intelligence (pp. 11–35). Cambridge, MA, USA: MIT Press.
- Wiener, N. (1964). *Extrapolation, interpolation, and smoothing of stationary time series*. The MIT Press.

Chapter 2

Probabilistic Modeling



This chapter introduces probabilistic modeling and reviews foundational concepts in Bayesian econometrics such as Bayesian inference, model selection, online learning, and Bayesian model averaging. We then develop more versatile representations of complex data with probabilistic graphical models such as mixture models.

1 Introduction

Not only is statistical inference from data intrinsically uncertain, but the type of data and relationships in the data that we seek to model are growing ever more complex. In this chapter, we turn to probabilistic modeling, a class of statistical models, which are broadly designed to characterize uncertainty and allow the expression of causality between variables. Probabilistic modeling is a meta-class of models, including generative modeling—a class of statistical inference models which maximizes the joint distribution, $p(X, Y)$, and Bayesian modeling, employing either maximum likelihood estimation or “fully Bayesian” inference. Probabilistic graphical models put the emphasis on causal modeling to simplify statistical inference of parameters from data. This chapter shall focus on the constructs of probabilistic modeling, as they relate to the application of both unsupervised and supervised machine learning in financial modeling.

While it seems natural to extend the previous chapters directly to a probabilistic neural network counterpart, it turns out that this does not develop the type of intuitive explanation of complex data that is needed in finance. It also turns out that neural networks are not a natural fit for probabilistic modeling. In other words, neural networks are well suited to pointwise estimation but lead to many difficulties in a probabilistic setting. In particular, they tend to be very data intensive—offsetting one of the major advantages of Bayesian modeling.

We will explore probabilistic modeling through the introduction of probabilistic graphical models—a data structure which is convenient for understanding the relationship between a multitude of different classes of models, both discriminative and generative. This representation will lead us neatly to Bayesian kernel learning, the subject of Chap. 3. We begin by introducing the reader to elementary topics in Bayesian modeling, which is a well-established approach for characterizing uncertainty in, for example, trading and risk modeling. Starting with simple probabilistic models of the data, we review some of the main constructs necessary to apply Bayesian methods in practice.

The application of probabilistic models to time series modeling, using filtering and hidden variables to dynamically represent the data is presented in Chap. 7.

Chapter Objectives

The key learning points of this chapter are:

- Apply Bayesian inference to data using simple probabilistic models;
- Understand how linear regression with probabilistic weights can be viewed as a simple probabilistic graphical model; and
- Develop more versatile representations of complex data with probabilistic graphical models such as mixture models and hidden Markov models.

Note that section headers ending with * are more mathematically advanced, often requiring some background in analysis and probability theory, and can be skipped by the less mathematically inclined reader.

2 Bayesian vs. Frequentist Estimation

Bayesian data analysis is distinctly different from classical (or “frequentist”) analysis in its treatment of probabilities, and in its resulting treatment of model parameters when compared to classical parametric analysis.¹

Bayesian analysts formulate probabilistic statements about uncertain events before collecting any additional evidence (i.e., “data”). These ex-ante probabilities (or, more generally, probability distributions plus underlying parameters) are called *priors*.

This notion of *subjective probabilities* is absent in classical estimation. In the classical world, all estimation and inference is based solely on observed data.

¹Throughout the first part of this chapter we will largely remain within the realm of parametric analysis. However, we shall later see examples of Bayesian methods for non- and semi-parametric modeling.

Both Bayesian and classical econometricians aim to learn more about a set of parameters, say θ . In the classical mindset, θ contains fixed but unknown elements, usually associated with an underlying population of interest (e.g., the mean and variance for credit card debt among US college students). Bayesians share with classicals the interest in θ and the definition of the population of interest.

However, they assign *ex ante* a prior probability to θ , labeled $p(\theta)$, which usually takes the form of a probability distribution with “known” moments. For example, Bayesians might state that the aforementioned debt amount has a normal distribution with mean \$3000 and standard deviation of \$1500. This prior may be based on previous research, related findings in the published literature, or it may be completely arbitrary. In any case, it is an inherently subjective construct.

Both schools then develop a theoretical framework that relates θ to observed data, say a “dependent variable” y , and a matrix of explanatory variables X . This relationship is formalized via a likelihood function, say $p(y | \theta, X)$ to stay with Bayesian notation. To stress, this likelihood function takes the exact same analytical form for both schools.

The classical analyst then collects a sample of observations from the underlying population of interest and, combining these data with the formulated statistical model, produces an estimate of θ , say $\hat{\theta}$. Any and all uncertainty surrounding the accuracy of this estimate is solely related to the notion that results are based on a sample, not data for the entire population. A different sample (of equal size) may produce slightly different estimates. Classicals express this uncertainty via “standard errors” assigned to each element of $\hat{\theta}$. They also have a strong focus on the behavior of $\hat{\theta}$ as the sample size increases. The behavior of estimators under increasing sample size falls under the heading of “asymptotic theory.”

The properties of most estimators in the classical world can only be assessed “asymptotically,” i.e. are only understood for the hypothetical case of an infinitely large sample. Also, virtually all specification tests used by frequentists hinge on asymptotic theory. This is a major limitation when the data size is finite.

Bayesians, in turn, combine prior and likelihood via Bayes’ rule to derive the *posterior distribution* of θ as

$$p(\theta | y, X) = \frac{p(\theta, y | X)}{p(y | X)} = \frac{p(\theta) p(y | \theta, X)}{p(y | X)} \propto p(\theta) p(y | \theta, X). \quad (2.1)$$

> Bayesian Modeling

Bayesian modeling is not about point estimation of a parameter value, θ , but rather updating and sharpening our subjective beliefs (our “prior”) about θ from the sample data. Thus, the sample data should contribute to “learning” about θ .

Bayesian Learning

Simply put, *the posterior distribution is just an updated version of the prior*. More specifically, the posterior is proportional to the prior multiplied by the likelihood. The likelihood carries all the current information about the parameters and the data. If the data has high informational content (i.e., allows for substantial learning about θ), the posterior will generally look very different from the prior. In most cases, it is much “tighter” (i.e., has a much smaller variance) than the prior. There is no room in Bayesian analysis for the classical notions of “sampling uncertainty,” and less a priori focus on the “asymptotic behavior” of estimators.²

Taking the Bayesian paradigm to its logical extreme, Duembgen and Rogers (2014) suggest to “estimate nothing.” They propose the replacement of the industry-standard estimation-based paradigm of calibration with an approach based on Bayesian techniques, wherein a posterior is iteratively obtained from a prior, namely stochastic filtering and MCMC. Calibration attempts to estimate, and then uses the estimates as if they were known true values—ignoring the estimation error. On the contrary, an approach based on a systematic application of the Bayesian principle is consistent: “There is never any doubt about what we should be doing to hedge or to mark-to-market a portfolio of derivatives, and whatever we do today will be consistent with what we did before, and with what we will do in the future.” Moreover, Bayesian model comparison methods enable one to easily compare models of very different types.

Marginal Likelihood

The term in the denominator of Eq. 2.1 is called the “marginal likelihood,” it is not a function of θ , and can usually be ignored for most components of Bayesian analysis. Thus, we usually work only with the numerator (i.e., prior times likelihood) for inference about θ . From Eq. 2.1 we know that this expression is proportional (“ \propto ”) to the actual posterior. However, the marginal likelihood is crucial for model comparison, so we will learn a few methods to derive it as a by-product of or following the actual posterior analysis. For some choices of prior and likelihood there exist analytical solutions for this term.

In summary, frequentists start with a “blank mind” regarding θ . They collect data to produce an estimate $\hat{\theta}$. They formalize the characteristics and uncertainty of $\hat{\theta}$ for a finite sample context (if possible) and a hypothetical large sample (asymptotic) case.

Bayesians collect data to *update a prior*, i.e. a pre-conceived probabilistic notion regarding θ .

²However, at times Bayesian analysis does rest on asymptotic results. Naturally, the general notion that a larger sample, i.e. more empirical information, is better than a small one also holds for Bayesian analysis.

3 Frequentist Inference from Data

Let us begin this section with a simple example which illustrates frequentist inference.

Example 2.1 Bernoulli Trials Example

Consider an experiment consisting of a single coin flip. We set the random variable Y to 0 if tails come up and 1 if heads come up. Then the probability density of Y is given by

$$p(y | \theta) = \theta^y(1 - \theta)^{1-y},$$

where $\theta \in [0, 1]$ is the probability of heads showing up.

You will recognize Y as a *Bernoulli random variable*. We view p as a function of y , but parameterized by the given parameter θ , hence the notation, $p(y | \theta)$.

More generally, suppose that we perform n such independent experiments (tosses) on the same coin. Denote these n realizations of Y as

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \in \{0, 1\}^n,$$

where, for $1 \leq i \leq n$, y_i is the result of the i th toss. What is the probability density of \mathbf{y} ?

Since the coin tosses are independent, the probability density of \mathbf{y} , i.e. the joint probability density of y_1, y_2, \dots, y_n , is given by the product rule

$$p(\mathbf{y} | \theta) = p(y_1, y_2, \dots, y_n | \theta) = \prod_{i=1}^n \theta^{y_i}(1 - \theta)^{1-y_i} = \theta^{\sum y_i}(1 - \theta)^{n - \sum y_i}.$$

Suppose that we have tossed the coin $n = 50$ times (performed $n = 50$ Bernoulli trials) and recorded the results of the trials as

0	0	1	0	0	1	0	0	0	0	0	1	0	1	1	1	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	0	

How can we estimate θ given these data?

Both the frequentists and Bayesians regard the density $p(\mathbf{y} | \theta)$ as a *likelihood*. Bayesians maintain this notation, whereas frequentists reinterpret $p(\mathbf{y} | \theta)$, which is

a function of \mathbf{y} (given the parameters θ : in our case, there is a single parameter, so θ is univariate, but this does not have to be the case) as a function of θ (given the specific sample \mathbf{y}), and write

$$\mathcal{L}(\theta) := \mathcal{L}(\theta | \mathbf{y}) := p(\mathbf{y} | \theta).$$

Notice that we have merely reinterpreted this probability density, whereas its functional form remains the same, in our case:

$$\mathcal{L}(\theta) = \theta^{\sum y_i} (1 - \theta)^{n - \sum y_i}.$$

> Likelihood

Likelihood is one of the seminal ideas of the frequentist school. It was introduced by one of its founding fathers, Sir Ronald Aylmer Fisher: “What has now appeared is that the mathematical concept of probability is ... inadequate to express our mental confidence or [lack of confidence] in making ... inferences, and that the mathematical quantity which usually appears to be appropriate for measuring our order of preference among different possible populations does not in fact obey the laws of probability. To distinguish it from probability, I have used the term ‘likelihood’ to designate this quantity...”—R.A. Fisher, *Statistical Methods for Research Workers*.

It is generally more convenient to work with the log of likelihood—the *log-likelihood*. Since \ln is a monotonically increasing function of its argument, the same values of θ maximize the log-likelihood as the ones that maximize the likelihood.

$$\ln \mathcal{L}(\theta) = \ln \left\{ \theta^{\sum y_i} (1 - \theta)^{n - \sum y_i} \right\} = \left(\sum y_i \right) \ln \theta + \left(n - \sum y_i \right) \ln(1 - \theta).$$

In order to find the value of θ that maximizes this expression, we differentiate with respect to θ and solve for the value of θ that sets the (partial) derivative to zero.

$$\frac{\partial}{\partial \theta} \ln \mathcal{L}(\theta) = \frac{\sum y_i}{\theta} + \frac{n - \sum y_i}{\theta - 1}.$$

Equating this to zero and solving for θ , we obtain the *maximum likelihood estimate* for θ :

$$\hat{\theta}_{\text{ML}} = \frac{\sum y_i}{n}.$$

To confirm that this value does indeed *maximize* the log-likelihood, we take the second derivative with respect to θ ,

$$\frac{\partial^2}{\partial \theta^2} \ln \mathcal{L}(\theta) = -\frac{\sum y_i}{\theta^2} - \frac{n - \sum y_i}{(\theta - 1)^2} < 0.$$

Since this quantity is strictly negative for all $0 < \theta < 1$, it is negative at $\hat{\theta}_{\text{ML}}$, and we do indeed have a maximum.

Example 2.2 Bernoulli Trials Example (continued)

Note that $\hat{\theta}_{\text{ML}}$ depends only on the sum of y_i s, we can answer our question: if in a sequence of 50 coin tosses exactly twelve heads come up, then

$$\hat{\theta}_{\text{ML}} = \frac{\sum y_i}{n} = \frac{12}{50} = 0.24.$$

A frequentist approach gives at a *single* value (a single “point”) as our estimate, 0.24—in this sense we are performing *point estimation*. When we apply a Bayesian approach to the same problem, we shall see that the Bayesian estimate is a probability distribution, rather than a single point.

Despite some mathematical formalism, the answer is intuitively obvious. If we toss a coin fifty times, and out of those twelve times it lands with heads up, it is natural to estimate the probability of getting heads as $\frac{12}{50}$. It is encouraging that the result of our calculation agrees with our intuition and common sense.

4 Assessing the Quality of Our Estimator: Bias and Variance

When we obtained our maximum likelihood estimate, we plugged in a specific number for $\sum y_i$, 12. In this sense the estimator is an ordinary function. However, we could also view it as a function of the *random* sample,

$$\hat{\theta}_{\text{ML}} = \frac{\sum Y_i}{n},$$

each Y_i being a random variable. A function of a random variable is itself a random variable, so we can compute its expectation and variance.

In particular, an expectation of the *error*

$$\mathbf{e} = \hat{\boldsymbol{\theta}} - \boldsymbol{\theta}$$

is known as *bias*,

$$\text{bias}(\hat{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \mathbb{E}(\mathbf{e}) = \mathbb{E}[\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}] = \mathbb{E}[\hat{\boldsymbol{\theta}}] - \mathbb{E}[\boldsymbol{\theta}].$$

As frequentists, we view the true value of $\boldsymbol{\theta}$ as a single, deterministic, fixed point, so we take it outside of the expectation:

$$\text{bias}(\hat{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \mathbb{E}[\hat{\boldsymbol{\theta}}] - \boldsymbol{\theta}.$$

In our case it is

$$\begin{aligned}\mathbb{E}[\hat{\theta}_{\text{ML}} - \theta] &= \mathbb{E}[\hat{\theta}_{\text{ML}}] - \theta = \mathbb{E}\left[\frac{\sum Y_i}{n}\right] - \theta = \frac{1}{n} \sum \mathbb{E}[Y_i] - \theta \\ &= \frac{1}{n} \cdot n(\theta \cdot 1 + (1 - \theta) \cdot 0) - \theta = 0,\end{aligned}$$

we see that the bias is zero, so this particular maximum likelihood estimator is *unbiased* (otherwise it would be *biased*).

What about the variance of this estimator?

$$\text{Var}[\hat{\theta}_{\text{ML}}] = \text{Var}\left[\frac{\sum Y_i}{n}\right] \stackrel{\text{independence}}{=} \frac{1}{n^2} \sum \text{Var}[Y_i] = \frac{1}{n^2} \cdot n \cdot \theta(1 - \theta) = \frac{1}{n} \theta(1 - \theta),$$

and we see that the variance of the estimator depends on the *true* value of θ .

For multivariate $\boldsymbol{\theta}$, it is useful to examine the *error covariance matrix* given by

$$\mathbf{P} = \mathbb{E}[\mathbf{e}\mathbf{e}^\top] = \mathbb{E}[(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta})(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta})^\top].$$

When estimating $\boldsymbol{\theta}$, our goal is to minimize the estimation error. This error can be expressed using loss functions. Supposing our parameter vector $\boldsymbol{\theta}$ takes values on some space Θ , a *loss function* $L(\boldsymbol{\theta})$ is a mapping from $\Theta \times \Theta$ into \mathbb{R} which quantifies the “loss” incurred by estimating $\boldsymbol{\theta}$ with $\hat{\boldsymbol{\theta}}$.

We have already seen loss functions in earlier chapters, but we shall restate the definitions here for completeness. One frequently used loss function is the *absolute error*,

$$L_1(\hat{\boldsymbol{\theta}}, \boldsymbol{\theta}) := \|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}\|_2 = \sqrt{(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta})^\top (\hat{\boldsymbol{\theta}} - \boldsymbol{\theta})},$$

where $\|\cdot\|_2$ is the Euclidean norm (it coincides with the absolute value when $\Theta \subseteq \mathbb{R}$). One advantage of the absolute error is that it has the same units as $\boldsymbol{\theta}$.

We use the *squared error* perhaps even more frequently than the *absolute error*:

$$L_2(\hat{\boldsymbol{\theta}}, \boldsymbol{\theta}) := \|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}\|_2^2 = (\hat{\boldsymbol{\theta}} - \boldsymbol{\theta})^\top (\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}).$$

While the squared error has the disadvantage compared to the absolute error of being expressed in *quadratic* units of $\hat{\theta}$, rather than the units of θ , it does not contain the cumbersome $\sqrt{\cdot}$ and is therefore easier to deal with mathematically.

The expected value of a loss function is known as the *statistical risk* of the estimator.

The statistical risks corresponding to the above loss functions are, respectively, the *mean absolute error*,

$$\text{MAE}(\hat{\theta}, \theta) := R_1(\hat{\theta}, \theta) := \mathbb{E} [L_1(\hat{\theta}, \theta)] := \mathbb{E} [\|\hat{\theta} - \theta\|_2] = \mathbb{E} [\sqrt{(\hat{\theta} - \theta)^\top (\hat{\theta} - \theta)}],$$

and, by far the most commonly used, *mean squared error (MSE)*,

$$\text{MSE}(\hat{\theta}, \theta) := R_2(\hat{\theta}, \theta) := \mathbb{E} [L_2(\hat{\theta}, \theta)] := \mathbb{E} [\|\hat{\theta} - \theta\|_2^2] = \mathbb{E} [(\hat{\theta} - \theta)^\top (\hat{\theta} - \theta)].$$

The square root of the mean squared error is called the *root mean squared error (RMSE)*. The *minimum mean squared error (MMSE)* estimator is the estimator that minimizes the mean squared error.

5 The Bias–Variance Tradeoff (Dilemma) for Estimators

It can easily be shown that the mean squared error separates into a variance and bias term:

$$\text{MSE}(\hat{\theta}, \theta) = \text{trVar}[\hat{\theta}] + \|\text{bias}(\hat{\theta}, \theta)\|_2^2,$$

where $\text{tr}(\cdot)$ is the trace operator. In the case of a scalar θ , this expression simplifies to

$$\text{MSE}(\hat{\theta}, \theta) = \text{Var}[\hat{\theta}] + \text{bias}(\hat{\theta}, \theta)^2.$$

In other words, the MSE is equal to the sum of the variance of the estimator and the squared bias.

The *bias–variance tradeoff* or *bias–variance dilemma* consists in the need to minimize these two sources of error, the variance and bias of an estimator, in order to minimize the mean squared error. Sometimes there is a tradeoff between minimizing bias and minimizing variance to achieve the least possible MSE. The concept of a bias–variance tradeoff in machine learning will be revisited in Chap. 4, within the context of statistical learning theory.

6 Bayesian Inference from Data

As before, let θ be the parameter of some statistical model and let $\mathbf{y} = y_1, \dots, y_n$ be n i.i.d. observations of some random variable Y . We capture our subjective assumptions about the model parameter θ , before observing the data, in the form of a prior probability distribution $p(\theta)$. Bayes' rule converts a prior probability into a posterior probability by incorporating the *evidence* provided by the observed data:

$$p(\theta | \mathbf{y}) = \frac{p(\mathbf{y} | \theta)}{p(\mathbf{y})} p(\theta)$$

allows us to evaluate the uncertainty in θ after we have observed \mathbf{y} . This uncertainty is characterized by the posterior probability $p(\theta | \mathbf{y})$. The effect of the observed data is expressed through $p(\mathbf{y} | \theta)$ —a function of θ referred to as the *likelihood function*. It expresses how likely the observed dataset was generated by a model with parameter θ .

Let us summarize some of the notation that will be important:

- The prior is $p(\theta)$;
- The likelihood is $p(\mathbf{y} | \theta) = \prod_{i=1}^n p(y_i | \theta)$, since the data is i.i.d.;
- The marginal likelihood $p(\mathbf{y}) = \int p(\mathbf{y} | \theta) p(\theta) d\theta$ is the likelihood with the dependency on θ marginalized out; and
- The posterior is $p(\theta | \mathbf{y})$.

➤ Bayesian Inference

Informally, Bayesian inference involves the following steps:

1. Formulate your statistical model as a collection of probability distributions conditional on different values for a parameter θ , about which you wish to learn;
2. Organize your beliefs about θ into a (prior) probability distribution;
3. Collect the data and insert them into the family of distributions given in Step 1;
4. Use Bayes' rule to calculate your new beliefs about θ ; and
5. Criticize your model and revise your modeling assumptions.

The following example shall illustrate the application of Bayesian inference for the Bernoulli parameter θ .

Example 2.3 Bernoulli Trials Example (continued)

θ is a probability, so it is bounded and must belong to the interval $[0, 1]$. We could assume that all values of θ in $[0, 1]$ are equally likely. Thus our prior could be that θ is uniformly distributed on $[0, 1]$, i.e. $\theta \sim \mathcal{U}(a = 0, b = 1)$.

This assumption would constitute an application of *Laplace's principle of indifference*, also known as the *principle of insufficient reason*: when faced with multiple possibilities, whose probabilities are unknown, assume that the probabilities of all possibilities are equal.

In the context of Bayesian estimation, applying Laplace's principle of indifference constitutes what is known as an *uninformative prior*. Our goal is, however, not to rely too much on the prior, but use the likelihood to proceed to a posterior based on new information.

The pdf of the uniform distribution, $\mathcal{U}(a, b)$, is given by

$$p(\theta) = \frac{1}{b - a}$$

if $\theta \in [a, b]$ and zero elsewhere. In our case, $a = 0, b = 1$, and so our uninformative uniform prior is given by

$$p(\theta) = 1, \quad \forall \theta \in [0, 1].$$

Let us derive the posterior based on this prior assumption. Bayes' theorem tells us that

$$\text{posterior} \propto \text{likelihood} \cdot \text{prior},$$

where \propto stands for “proportional to,” so the left- and right-hand side are equal up to a normalizing constant which depends on the data but not on θ . The posterior is

$$p(\theta | x_{1:n}) \propto p(x_{1:n} | \theta) p(\theta) = \theta^{\sum x_i} (1 - \theta)^{n - \sum x_i} \cdot 1.$$

If the prior is uniform, i.e. $p(\theta) = 1$, then after $n = 5$ trials we see from the data that

$$p(\theta | x_{1:n}) \propto \theta(1 - \theta)^4. \tag{2.2}$$

After 10 trials we have

(continued)

Example 2.3 (continued)

$$p(\theta | x_{1:n}) \propto \theta(1-\theta)^4 \times \theta(1-\theta)^4 = \theta^2(1-\theta)^8. \quad (2.3)$$

From the shape of the resulting pdf, we recognize it as the pdf of the Beta distribution^a

$$\text{Beta}\left(\theta | \sum x_i, n - \sum x_i\right),$$

and we immediately know that the missing normalizing constant factor is

$$\frac{1}{B\left(\sum x_i, n - \sum x_i\right)} = \frac{\Gamma\left(\sum x_i\right) \Gamma\left(n - \sum x_i\right)}{\Gamma(n)}.$$

Let us now assume that we have tossed the coin fifty times and, out of those fifty coin tosses, we get heads on twelve. Then our posterior distribution becomes

$$\theta | x_{1:n} \sim \text{Beta}(\theta | 12, 38).$$

Then, from the properties of this distribution,

$$\mathbb{E}[\theta | x_{1:n}] = \frac{\sum x_i}{\sum x_i + (n - \sum x_i)} = \frac{\sum x_i}{n} = \frac{12}{12 + 38} = \frac{12}{50} = 0.24,$$

$$\text{Var}[\theta | x_{1:n}] = \frac{(\sum x_i)(n - \sum x_i)}{(\sum x_i + n - \sum x_i)^2 (\sum x_i + n - \sum x_i + 1)} \quad (2.4)$$

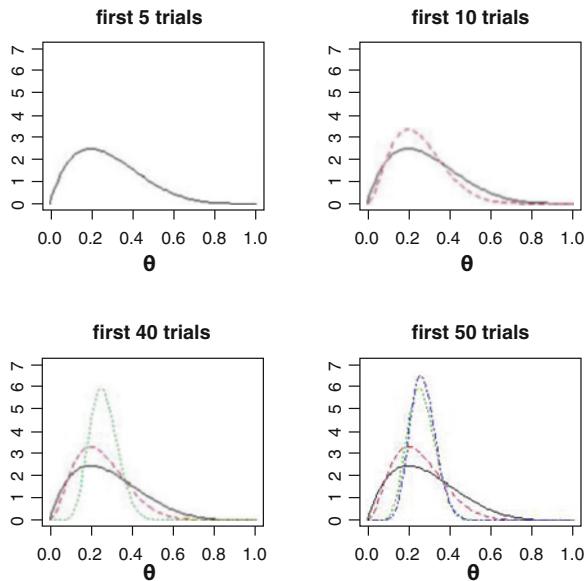
$$= \frac{n \sum x_i - (\sum x_i)^2}{n^2(n+1)} = \frac{12 \cdot 38}{(12+38)^2(12+38+1)} = \frac{456}{127500} = 0.00357647058. \quad (2.5)$$

The standard deviation being, in units of probability, $\sqrt{\frac{456}{127500}} = 0.05980360012$.

Notice that the mean of the posterior, 0.24, matches the frequentist maximum likelihood estimate of θ , $\hat{\theta}_{\text{ML}}$, and our intuition. Again, it is not unreasonable to assume that the probability of getting heads is 0.24 if we observe heads on twelve out of fifty coin tosses.

^aThe function's argument is now θ , not x_i , so it is not the pdf of a Bernoulli distribution.

Fig. 2.1 The posterior distribution of θ against successive numbers of trials. The x-axis shows the values of theta. The shape of the distribution tightens as the Bayesian model is observed to “learn”



Note that we did not need to evaluate the marginal likelihood in the example above, only the θ dependent terms were evaluated for the purpose of the plot. Thus each plot in Fig. 2.1 is only representative of the posterior up to a scaling.

! The Principle of Indifference

In practice, the principle of indifference should be used with great care, as we are assuming a property of the data strictly greater than we know. Saying “the probabilities of the outcomes are equally likely” contains strictly more information than “I don’t know what the probabilities of the outcomes are.”

If someone tosses a coin and then covers it with her hand, asking you, “heads or tails?” it is probably relatively sensible to assume that the two possibilities are equally likely, effectively assuming that the coin is unbiased.

If an investor asks you, “Will the stock price of XYZ increase?” you should think twice before applying Laplace’s principle of indifference and replying “Well, there is a 50% chance that XYZ will grow, you can either long or short XYZ.” Clearly there are other important considerations such as the amount by which the stock could increase versus decrease, limits on portfolio exposure to market risk factors, and anticipation of other market events such as earnings announcements. In other words, the implications of going long or short will not necessarily be equal.

6.1 A More Informative Prior: The Beta Distribution

Continuing with the above example, let us question our prior. Is it somewhat *too* uninformative? After all, most coins in the world are probably close to being unbiased. We could use a $\text{Beta}(\alpha, \beta)$ prior instead of the Uniform prior. Picking $\alpha = \beta = 2$, for example, will give a distribution on $[0, 1]$ centered on $\frac{1}{2}$, incorporating a prior assumption that the coin is unbiased.

The pdf of this prior is given by

$$p(\theta) = \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1}, \forall \theta \in [0, 1],$$

and so the posterior becomes

$$\begin{aligned} p(\theta | x_{1:n}) &\propto p(x_{1:n} | \theta) p(\theta) \\ &= \theta^{\sum x_i} (1-\theta)^{n-\sum x_i} \cdot \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1} \propto \theta^{(\alpha+\sum x_i)-1} (1-\theta)^{(\beta+n-\sum x_i)-1}, \end{aligned}$$

which we recognize as a pdf of the distribution

$$\text{Beta}\left(\theta | \alpha + \sum x_i, \beta + n - \sum x_i\right).$$

Why did we pick this prior distribution? One reason is that its pdf is defined over the compact interval $[0, 1]$, unlike, for example, the normal distribution, which has tails extending to $-\infty$ and $+\infty$. Another reason is that we are able to choose parameters which center the pdf at $\theta = \frac{1}{2}$, incorporating the prior assumption that the coin is unbiased.

If we initially assume a $\text{Beta}(\theta | \alpha = 2, \beta = 2)$ prior, then the posterior expectation is

$$\begin{aligned} \mathbb{E}[\theta | x_{1:n}] &= \frac{\alpha + \sum x_i}{\alpha + \sum x_i + \beta + n - \sum x_i} = \frac{\alpha + \sum x_i}{\alpha + \beta + n} \\ &= \frac{2 + 12}{2 + 2 + 50} = \frac{7}{27} \approx 0.259. \end{aligned}$$

Notice that both the prior and posterior belong to the same probability distribution family. In Bayesian estimation theory we refer to such prior and posterior as *conjugate distributions* (with respect to this particular likelihood function).

Unsurprisingly, since now our prior assumption is that the coin is unbiased, $\frac{12}{50} < \mathbb{E}[\theta | x_{1:n}] < \frac{1}{2}$.

Perhaps surprisingly, we are also somewhat more certain about the posterior (its variance is smaller) than when we assumed the uniform prior.

Notice that the results of Bayesian estimation are sensitive—to varying degree in each specific case—to the choice of prior distribution:

$$p(\theta | \alpha, \beta) = \frac{(\alpha + \beta - 1)!}{(\alpha - 1)!(\beta - 1)!} \theta^{\alpha-1} (1 - \theta)^{\beta-1} = \Gamma(\alpha, \beta) \theta^{\alpha-1} (1 - \theta)^{\beta-1}. \quad (2.6)$$

So for the above example, this marginal likelihood would be evaluated with $\alpha = 13$ and $\beta = 39$ since there are 12 observed 1s and 38 observed 0s.

6.2 Sequential Bayesian updates

In the previous section we saw that, starting with the prior

$$\text{Beta}(\theta | \alpha, \beta),$$

we arrived at the Beta-distributed posterior,

$$\text{Beta}\left(\theta | \alpha + \sum x_i, \beta + n - \sum x_i\right).$$

What would happen if, instead of observing all twelve coin tosses at once, we (i) considered each coin toss in turn; (ii) obtained our posterior; and (iii) used that posterior as a prior for an update based on the information from the next coin toss?

The above two formulae give the answer to this question. We start with our initial prior,

$$\text{Beta}(\theta | \alpha, \beta),$$

then, substituting $n = 1$ into the second formula, we get

$$\text{Beta}(\theta | \alpha + x_1, \beta + 1 - x_1).$$

Using this posterior as a prior before the second coin toss, we obtain the next posterior as

$$\text{Beta}(\theta | \alpha + x_1 + x_2, \beta + 2 - x_1 - x_2).$$

Proceeding along these lines, after all ten coin tosses, we end up with

$$\text{Beta}\left(\theta | \alpha + \sum x_i, \beta + n - \sum x_i\right),$$

the same result that we would have attained if we processed all ten coin tosses as a single “batch,” as we did in the previous section.

This insight forms the basis for a *sequential* or *iterative* application of Bayes’ theorem—sequential Bayesian updates—the foundation for real-time *Bayesian filtering*. In machine learning, this mechanism for updating our beliefs in response to new data is referred to as “online learning.”

6.2.1 Online Learning

An important aspect of Bayesian learning is the capacity to update the posterior in response to the arrival of new data, \mathbf{y}' . The posterior over \mathbf{y} now becomes the prior, and the new posterior is updated to

$$p(\theta | \mathbf{y}', \mathbf{y}) = \frac{p(\mathbf{y}' | \theta) p(\theta | \mathbf{y})}{\int_{\theta \in \Theta} p(\mathbf{y}' | \theta) p(\theta | \mathbf{y}) d\theta}. \quad (2.7)$$

6.2.2 Prediction

In auto-correlated data, often encountered in financial econometrics, it is common to use Bayesian models for prediction. We can write that the density of the new predicted value y' given the previous data y is the expected value of the likelihood of the new data under the posterior density $p(\theta | y)$:

$$p(y' | y) = \mathbb{E}_{\theta | y}[p(y' | y, \theta)] = \int_{\theta \in \Theta} p(y' | y, \theta) p(\theta | y) d\theta. \quad (2.8)$$

? Multiple Choice Question 1

Which of the following statements are true:

1. A frequentist performs statistical inference by finding the best fit parameters. The Bayesian finds the distribution of the parameters assuming a prior.
2. Frequentist inference can be regarded as a special case of Bayesian inference when the prior is a Dirac delta-function.
3. Bayesian inference is well suited to online learning, an experimental design under which the model is continuously updated as new data arrives.
4. Prediction, under Bayesian inference, is the conditional expectation of the predicted variable under the posterior distribution of the parameter.

6.3 Practical Implications of Choosing a Classical or Bayesian Estimation Framework

If the sample size is large and the likelihood function “well-behaved” (which usually means a simple function with a clear maximum, plus a small dimension for θ), classical and Bayesian analysis are essentially on the same footing and will produce virtually identical results. This is because the likelihood function and empirical data will dominate any prior assumptions in the Bayesian approach.

If the sample size is large but the dimensionality of θ is high and the likelihood function is less tractable (which usually means highly non-linear, with local maxima, flat spots, etc.), a Bayesian approach may be preferable purely from a computational standpoint. It can be very difficult to attain reliable estimates via maximum likelihood estimation (MLE) techniques, but it is usually straightforward to derive a posterior distribution for the parameters of interest using Bayesian estimation approaches, which often operate via sequential draws from known distributions.

If the sample size is small, Bayesian analysis can have substantial advantages over a classical approach. First, Bayesian results do not depend on asymptotic theory to hold for their interpretability. Second, the Bayesian approach combines the sparse data with subjective priors. Well-informed priors can increase the accuracy and efficiency of the model. Conversely, of course, poorly chosen priors³ can produce misleading posterior inference in this case. Thus, under small sample conditions, the choice between Bayesian and classical estimation often distills to a choice between trusting the asymptotic properties of estimators and trusting one’s priors.

7 Model Selection

Beyond the inference challenges described above, there are a number of problems with the classical approach to model selection which Bayesian statistics solves. For example, it has been shown by Breiman (2001) that the following three linear regression models have a residual sum of squares (RSS) which are all within 1%:

$$\text{Model 1} \quad \hat{Y} = 2.1 + 3.8X_3 - 0.6X_8 + 83.2X_{13} - 2.1X_{17} + 3.2X_{27}, \quad (2.9)$$

$$\text{Model 2} \quad \hat{Y} = -8.9 + 4.6X_5 + 0.01X_6 + 12.0X_{15} + 17.5X_{21} + 0.2X_{22}, \quad (2.10)$$

$$\text{Model 3} \quad \hat{Y} = -76.7 + 9.3X_2 + 22.0X_7 - 13.2X_8 + 3.4X_{11} + 7.2X_{28}. \quad (2.11)$$

³For example, priors that place substantial probability mass on practically infeasible ranges of θ —this often happens inadvertently when parameter transformations are involved in the analysis.

You could, for example, think of each model being used to find the fair price of an asset Y , where each X_i are the contemporaneous (i.e., measured at the same time) firm characteristics.

- Which model is better?
- How would your interpretation of which variables are the most important change between models?
- Would you arrive at different conclusions about the market signals if you picked, say, Model 1 versus Model 2?
- How would you eliminate some of the ambiguity resulting from this outcome of statistical inference?

Of course one direction is to simply analyze the F-scores of each independent variable and select the model which has the most statistically significant fitted coefficients. But this is unlikely to reliably discern the models when the fitted coefficients are comparable in statistical significance.

It is well known that the goodness-of-fit measures, such as RSS's and F-scores, do not scale well to more complex datasets where there are several independent variables. This leads to modelers drawing different conclusions about the same data, and is famously known as the “Rashomon effect.” Yet many studies and models in finance are still built this way and make use of information criterion and regularization techniques such as Akaike’s information criteria (AIC).

A limitation for more robust frequentist model comparison is the requirement that the models being compared are “nested.” That is, one model should be a subset of the other model being compared, e.g.

$$\text{Model 1} \quad \hat{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \quad (2.12)$$

$$\text{Model 2} \quad \hat{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{11} X_1^2. \quad (2.13)$$

Model 1 is nested in Model 2 and we refer to the model selection as a “nested model selection.” In contrast to classical model selection, Bayesian model selection need not be restricted to nested models.

7.1 Bayesian Inference

We now consider the more general setting—selection and updating of several candidate models in response to a dataset \mathbf{y} . The “model” can be any data model, not just a regression, and the notation used here reflects that. In Bayesian inference, a model is a family of probability distributions, each of which can explain the observed data. More precisely, a model \mathcal{M} is the set of likelihoods $p(\mathbf{x}_n | \boldsymbol{\theta})$ over all possible parameter values Θ .

For example, consider the case of flipping a coin n times with an unknown bias $\theta \in \Theta \equiv [0, 1]$. The data $\mathbf{x}_n = \{x_i\}_{i=1}^n$ is now i.i.d. Bernoulli and if we observe the number of heads $X = x$, the model is the family of binomial distributions

$$\mathcal{M} := \{\mathbb{P}[X = x | n, \theta] = \binom{n}{x} \theta^x (1 - \theta)^{n-x}\}_{\theta \in \Theta}. \quad (2.14)$$

Each one of these distributions is a potential explanation of the observed head count x . In the Bayesian method, we maintain a belief over which elements in the model are considered plausible by reasoning about $p(\boldsymbol{\theta} | \mathbf{x}_n)$. See Example 1.1 for further details of this experiment.

We start by re-writing the Bayesian inference formula with explicit inclusion of model indexes. You will see that we have dropped \mathbf{X} since the exact composition of explanatory data is implicitly covered by model index \mathcal{M}_i :

$$p(\boldsymbol{\theta}_i | \mathbf{x}_n, \mathcal{M}_i) = \frac{p(\boldsymbol{\theta}_i | \mathcal{M}_i) p(\mathbf{x}_n | \boldsymbol{\theta}_i, \mathcal{M}_i)}{p(\mathbf{x}_n | \mathcal{M}_i)} \quad i = 1, 2. \quad (2.15)$$

This expression shows that differences across models can occur due to differing priors for $\boldsymbol{\theta}$ and/or differences in the likelihood function. The marginal likelihood in the denominator will usually also differ across models.

7.2 Model Selection

So far, we just considered parameter inference when the model has already been selected. The Bayesian setting offers a very flexible framework for the comparison of competing models—this is formally referred to as “model selection.” The models do not have to be nested—all that is required is that the competing specifications share the same \mathbf{x}_n .

Suppose there are two models, denoted \mathcal{M}_1 and \mathcal{M}_2 , each associated with a respective set of parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. We seek the most “probable” model given the observed data \mathbf{x}_n . We first apply Bayes’ rule to derive an expression for the *posterior model probability*

$$p(\mathcal{M}_i | \mathbf{x}_n) = \frac{p(\mathcal{M}_i) p(\mathbf{x}_n | \mathcal{M}_i)}{\sum_j p(\mathbf{x}_n | \mathcal{M}_j) p(\mathcal{M}_j)} \quad i = 1, 2. \quad (2.16)$$

Here $p(\mathcal{M}_i)$ is a prior distribution over models that we have selected; a common practice is to set this to a uniform distribution over the models. The value $p(\mathbf{x}_n | \mathcal{M}_i)$ is a marginal likelihood function—a likelihood function over the space of models in which the parameters have been marginalized out:

$$p(\mathbf{x}_n \mid \mathcal{M}_i) = \int_{\boldsymbol{\theta}_i \in \Theta_i} p(\mathbf{x}_n \mid \boldsymbol{\theta}_i, \mathcal{M}_i) p(\boldsymbol{\theta}_i \mid \mathcal{M}_i) d\boldsymbol{\theta}_i. \quad (2.17)$$

From a sampling perspective, this marginal likelihood can be interpreted as the probability that the model could have generated the observed data, under the chosen prior belief over its parameters. More precisely, the marginal likelihood can be viewed as the probability of generating \mathbf{x}_n from a model \mathcal{M}_i whose parameters $\boldsymbol{\theta}_i \in \Theta_i$ are sampled at random from the prior $p(\boldsymbol{\theta}_i \mid \mathcal{M}_i)$. For this reason, it is often referred to here as the *model evidence* and plays an important role in model selection that we will see later.

We can now construct the *posterior odds ratio* for the two models as

$$\frac{p(\mathcal{M}_1 \mid \mathbf{x}_n)}{p(\mathcal{M}_2 \mid \mathbf{x}_n)} = \frac{p(\mathcal{M}_1) p(\mathbf{x}_n \mid \mathcal{M}_1)}{p(\mathcal{M}_2) p(\mathbf{x}_n \mid \mathcal{M}_2)}, \quad (2.18)$$

which is simply the prior odds multiplied by the ratio of the evidence for each model.

Under equal model priors (i.e., $p(\mathcal{M}_1) = p(\mathcal{M}_2)$) this reduces to the *Bayes' factor* for Model 1 vs. 2, i.e.

$$B_{1,2} = \frac{p(\mathbf{x}_n \mid \mathcal{M}_1)}{p(\mathbf{x}_n \mid \mathcal{M}_2)}, \quad (2.19)$$

which is simply the ratio of marginal likelihoods for the two models. Since Bayes' factors can become quite large, we usually prefer to work with its logged version

$$\log B_{1,2} = \log p(\mathbf{x}_n \mid \mathcal{M}_1) - \log p(\mathbf{x}_n \mid \mathcal{M}_2). \quad (2.20)$$

The derivation of BFs and thus model comparison is straightforward if expressions for marginal likelihoods are analytically known or can be easily derived. However, often this can be quite tricky, and we will learn a few techniques to compute marginal likelihoods in this book.

7.3 Model Selection When There Are Many Models

Suppose now that a set of models $\{\mathcal{M}_i\}$ may be used to explain the data \mathbf{x}_n . $\boldsymbol{\theta}_i$ represents the parameters of model \mathcal{M}_i . Which model is “best”?

We answer this question by estimating the posterior distribution over models:

$$p(\mathcal{M}_i \mid \mathbf{x}_n) = \frac{\int_{\boldsymbol{\theta}_i \in \Theta_i} p(\mathbf{x}_n \mid \boldsymbol{\theta}_i, \mathcal{M}_i) p(\boldsymbol{\theta}_i \mid \mathcal{M}_i) d\boldsymbol{\theta}_i p(\mathcal{M}_i)}{\sum_j p(\mathbf{x}_n \mid \mathcal{M}_j) p(\mathcal{M}_j)}. \quad (2.21)$$

Table 2.1 Jeffreys' scale is used to assess the comparative strength of evidence in favor of one model over another

$ \ln B $	relative odds	favoured model's probability	Interpretation
< 1.0	< 3:1	< 0.750	not worth mentioning
< 2.5	< 12:1	0.923	weak
< 5.0	< 150:1	0.993	moderate
> 5.0	> 150:1	> 0.993	strong

As before we can compare any two models via the *posterior odds*, or if we assume equal priors, by the BFs. Model selection is always relative rather than absolute. We must always pick a reference model \mathcal{M}_2 and decide whether model \mathcal{M}_1 has more strength. We use Jeffreys' scale to assess the strength of evidence as shown in Table 2.1.

Example 2.4 Model Selection

You compare two models for explaining the behavior of a coin. The first model, \mathcal{M}_1 , assumes that the probability of a head is fixed to 0.5. Notice that this model does not have any parameters. The second model, \mathcal{M}_2 , assumes the probability of a head is set to an unknown $\theta \in \Theta = (0, 1)$ with a uniform prior on $\theta : p(\theta | \mathcal{M}_2) = 1$. For simplicity, we additionally choose a uniform model prior $p(\mathcal{M}_1) = p(\mathcal{M}_2)$.

Suppose we flip the coin $n = 200$ times and observe $X = 115$ heads. Which model should we prefer in light of this data? We compute the model evidence for each model. The model evidence for \mathcal{M}_1

$$p(x | \mathcal{M}_1) = \binom{n}{x} \frac{1}{2^{200}} \approx 0.005956. \quad (2.22)$$

The model evidence of \mathcal{M}_2 requires integrating over θ :

$$p(x | \mathcal{M}_2) = \int_0^1 p(x | \theta, \mathcal{M}_2) p(\theta | \mathcal{M}_2) d\theta \quad (2.23)$$

$$= \int_0^1 \binom{n}{x} \theta^{115} (1 - \theta)^{200-115} d\theta \quad (2.24)$$

$$= \frac{1}{201} \approx 0.004975. \quad (2.25)$$

(continued)

Example 2.4 (continued)

Note that we have used the definition of the Beta density function

$$p(\theta | \alpha, \beta) = \frac{(\alpha + \beta - 1)!}{(\alpha - 1)!(\beta - 1)!} \theta^{\alpha-1} (1 - \theta)^{\beta-1} \quad (2.26)$$

to evaluate the integral in the marginal density function above.

The Bayes' factor in favor of \mathcal{M}_1 is 1.2 and thus $|\ln B| = 0.182$ and there is no evidence in favor of \mathcal{M}_1 .

! Frequentist Approach

An interesting aside here is that a frequentist hypothesis test would reject the null hypothesis $\theta = 0.5$ at the $\alpha = 0.05$ level. The probability of generating at least 115 heads under model \mathcal{M}_1 is approximately 0.02. The probability of generating at least 115 tails is also 0.02. So a two-sided test would give a p-value of approximately 4%.

! Hyperparameters

We note in passing that the prior distribution in the example above does not involve any parameterization. If the prior is a parameterized distribution, then the parameters of the prior are referred to as *hyperparameters*. The distributions of the hyperparameters are known as *hyperpriors*. “Bayesian hierarchical modeling” is a statistical model written in multiple levels (hierarchical form) that estimates the parameters of the posterior distribution using the Bayesian method.

7.4 Occam's Razor

The model evidence performs a vital role in the prevention of model over-fitting. Models that are too simple are unlikely to generate the dataset. On the other hand, models that are too complex can generate many possible data sets, but they are unlikely to generate any particular dataset at random. Bayesian inference therefore automates the determination of model complexity using the training data x_n alone and does not need special “fixes” (a.k.a regularization and information criteria) to prevent over-fitting. The underlying philosophical principle of selecting the simplest model, if given a choice, is known as “Occam’s razor” (Fig. 2.2).

We maintain a belief over which parameters in the model we consider plausible by reasoning with the posterior

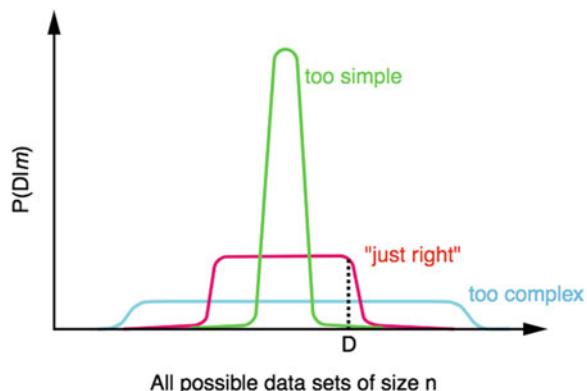
$$p(\theta_i | x_n, \mathcal{M}_i) = \frac{p(x_n | \theta_i, \mathcal{M}_i) p(\theta_i | \mathcal{M}_i)}{p(x_n | \mathcal{M}_i)}, \quad (2.27)$$

and we may choose the parameter value which maximizes the posterior distribution (MAP).

7.5 Model Averaging

Marginal likelihoods can also be used to derive *model weights* in *Bayesian model averaging (BMA)*. Informally, the intuition behind BMA is that we are never fully convinced that a single model is the correct one for our analysis at hand. There are usually several (and often millions of) competing specifications. To explicitly incorporate this notion of “model uncertainty,” one can estimate every model separately, compute relative probability weights for each model, and then generate model-averaged posterior distributions for the parameters (and predictions)

Fig. 2.2 The model evidence $p(D | m)$ performs a vital role in the prevention of model over-fitting. Models that are too simple are unlikely to generate the dataset. Models that are too complex can generate many possible data sets, but they are unlikely to generate any particular dataset at random. Source: Rasmussen and Ghahramani (2001)



of interest. We often choose BMA when there is not strong enough evidence for any particular model. Prediction of a new point y_* under BMA is given over m models as the weighted average

$$p(y_*|y) = \sum_{i=1}^m p(y_*|y, M_i)p(M_i|y). \quad (2.28)$$

Note that model-averaged prediction would be cumbersome to accomplish in a classical framework, and thus constitutes another advantage of employing a Bayesian estimation approach.

? Multiple Choice Question 2

Which of the following statements are true:

1. Bayesian inference is ideally suited to model selection because the model evidence effectively penalizes over-parameterized models.
 2. The principle of Occam's razor is to simply choose the model with the least bias.
 3. Bayesian model averaging uses the uncertainty from the model to weight the output from each model.
 4. Bayesian model averaging is a method of consensus voting between models—the best candidate model is selected for each new observation.
 5. Hierarchical Bayesian modeling involves nesting Bayesian models through parameterizations of prior distributions and their distributions.
-

8 Probabilistic Graphical Models

Graphical models (a.k.a. Bayesian networks) are a method for representing relationships between random variables in a probabilistic model. They provide a useful tool for big data, providing graphical representations of complex datasets.

To see how graphical models arise, we can revisit the familiar perceptron model from the previous chapter in a probabilistic framework, i.e. the network weights are now assumed to be probabilistic. As a starting point, consider a logistic regression classifier with probabilistic output:

$$\mathbb{P}[G | X] = \sigma(U) = \frac{1}{1 + e^{-U}}, \quad U = wX + b, \quad G \in \{0, 1\}, \quad X \in \mathbb{R}^p. \quad (2.29)$$

By Bayes' law, we know that the posterior probabilities must be given by the likelihood, prior and evidence:

$$\mathbb{P}[G | X] = \frac{\mathbb{P}[X | G]\mathbb{P}[G]}{\mathbb{P}[X]} = \frac{1}{1 + e^{-(\log(\frac{\mathbb{P}[X | G]}{\mathbb{P}[X | G^c]})) + \log(\frac{\mathbb{P}[G]}{\mathbb{P}[G^c]})}}, \quad (2.30)$$

where G^c is the complement of G . So the outputs are only posterior probabilities when the weights and biases are, respectively, likelihood and log-odds ratios:

$$w_j = \frac{\mathbb{P}[X_j | G]}{\mathbb{P}[X_j | G^c]}, \forall j \in \{1, \dots, p\}, \quad b = \log\left(\frac{\mathbb{P}[G]}{\mathbb{P}[G^c]}\right). \quad (2.31)$$

In particular, the X'_j 's must be *conditionally independent* over G ; otherwise, the outputs from the logistic regression are not the true posterior probabilities. Put differently, the posteriors of the input given the class can only be found when the input is mutually independent given the class G . In this case, the logistic regression is a naive Bayes' classifier—a type of generative model which models the joint distribution as the products of marginals, $\mathbb{P}[X, G] = \mathbb{P}[G] \prod_{j=1}^p \mathbb{P}[X_j | G]$. Hence, under this data assumption, logistic regression is the discriminative counterpart to naive Bayes. Figure 2.3b shows an example of an equivalent logistic regression models and naive Bayes' binary classifier for the case when the inputs are binary.

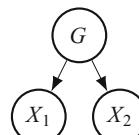
Furthermore, if the conditional density functions of the inputs, $\mathbb{P}[X_j | G]$, are Gaussian (but not necessarily identical), then we can establish equivalence between logistic regression and a Gaussian naive Bayes' classifier. See Exercise 2.7 for establishing the equivalence when the inputs are binary.

The graphical model captures the causal process (Pearl, 1988) by which the observed data was generated. For this reason, such models are often called generative models.

Fig. 2.3 Logistic regression
 $f_w(X) = \sigma(w^T X)$ and an
equivalent naive Bayes'
classifier. (a) Logistic
regression weights and
resulting predictions. (b) A
naive Bayes' classifier with
the same probabilistic output

	X_1	X_2	$F_w(X)$
	1	1	0.70
	1	0	0.74
	0	1	0.20
	0	0	0.24

(a)



$P_1(c)$	$G \ \mathbb{P}_1[X_1 G]$	$G \ \mathbb{P}_1[X_2 G]$
0.5	1	0.8
0	0	0.3

(b)

Naive Bayes' classification is the simplest form of a probabilistic graphical model (PGM) with a directed graph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$, where the edges, \mathcal{E} , represent the conditional dependence between the random variables $\mathcal{X} = (X, Y)$. For example, Fig. 2.3b shows the dependence of the response G on X in the naive Bayes' classifier. Such graphs, provided they are directed, are often referred to as "Bayesian networks." Such graphical model captures the causal process by which the observed data was generated. If the graph is undirected—an undirected graphical model (UGM), as in restricted Boltzmann machines (RBMs), then the network is referred to as a "Markov network" or Markov random field.

RBM have the specific restriction that there are no observed–observed and hidden–hidden node connections. RBMs are an example of continuous latent variable models which find the probability of an observed variable by marginalizing over the continuous latent variable, Z ,

$$p(x) = \int p(x | z) p(z) dz. \quad (2.32)$$

This type of graphical model corresponds to that of factor analysis. Other related types of graphical models include mixture models.

8.1 Mixture Models

A standard mixture probability density of a continuous and independently (but not identically) distributed random variable X , whose value is denoted by x , is defined as

$$p(x; \nu) = \sum_{k=1}^K \pi_k p(x; \theta_k). \quad (2.33)$$

The mixture density has K components (or states) and is defined by the parameter set $\nu = \{\theta, \pi\}$, where $\pi = \{\pi_1, \dots, \pi_K\}$ is the set of weights given to each component and $\theta = \{\theta_1, \dots, \theta_K\}$ is the set of parameters describing each component distribution. A well-known mixture model is the Gaussian mixture model (GMM):

$$p(x) = \sum_{k=1}^K \pi_k N(x; \mu_k, \sigma_k^2), \quad (2.34)$$

where each component parameter vector θ_k is the mean and variance parameters, μ_k and σ_k^2 .

When X is discrete, the graphical model is referred to as a “discrete mixture model.” Examples of GMMs are common in finance. Risk managers, for example, speak in terms of “correlation breakdowns,” “market contagion,” and “volatility shocks.” Finger (1997) presents a two-component GMM for modeling risk under normal and stressed market scenarios which has become standard methodology for stressed Value-at-Risk and Economic Capital modeling in the investment banking sector. Mixture models can also be used to cluster data and have a non-probabilistic analog called the K-means algorithm which is a well-known unsupervised learning method used in finance and other fields.

Before such a model can be fitted, it is necessary to introduce an additional variable which represents the current state of the data, i.e. which of the mixture component distributions is the current observation drawn from.

8.1.1 Hidden Indicator Variable Representation of Mixture Models

Let us first suppose that the independent random variable, X , has been observed over N data points, $\mathbf{x}_N = \{x_1, \dots, x_N\}$. The set is assumed to be generated by a K -component mixture model.

To indicate the mixture component from which a sample was drawn, we introduce an independent hidden (a.k.a. latent) discrete random variable, $S \in \{1, \dots, K\}$. For each observation x_i , the value of S is denoted as s_i , and is encoded as a binary vector of length K . We set the vector’s k -th component, $(s_i)_k = 1$ to indicate that the k -th mixture component is selected, while all other states are set to 0. As a consequence,

$$1 = \sum_{k=1}^K (s_i)_k. \quad (2.35)$$

We can now specify the joint probability distribution of X and S in terms of a marginal density $p(s_i; \boldsymbol{\pi})$ and a conditional density $p(x_i | s_i; \theta)$ as

$$p(\mathbf{x}_n, \mathbf{s}_n; \boldsymbol{\nu}) = \prod_i^N p(x_i | s_i; \theta) p(s_i; \boldsymbol{\pi}), \quad (2.36)$$

where the marginal densities $p(s_i; \boldsymbol{\pi})$ are drawn from a multinomial distribution that is parameterized by the mixing weights $\boldsymbol{\pi} = \{\pi_1, \dots, \pi_K\}$:

$$p(s_i; \boldsymbol{\pi}) = \prod_{k=1}^K \pi_k^{(s_i)_k}, \quad (2.37)$$

or, more simply,

$$\mathbb{P}[(s_i)_k = 1] = \pi_k. \quad (2.38)$$

Naturally the mixing weights, $\pi_k \in [0, 1]$, must satisfy

$$1 = \sum_{k=1}^K \pi_k. \quad (2.39)$$

8.1.2 Maximum Likelihood Estimation

The maximum likelihood method of estimating mixture models is known as the expectation–maximization (EM) algorithm. The goal of the EM is to maximize the likelihood of the data given the model, i.e. maximize

$$\mathcal{L}(\nu) = \log \left\{ \sum_s p(\mathbf{x}_n, \mathbf{s}_n; \nu) \right\} = \sum_{i=1}^N \sum_{k=1}^K (s_i)_k \log \{\pi_k p(x_i; \theta_k)\}. \quad (2.40)$$

If the sequence of states \mathbf{s}_n were known, then the estimation of the model parameters π, θ would be straightforward; conditioned on the state variables and the observations, Eq. 2.40 could be maximized with respect to the model parameters. However, the value of the state variable is unknown. This suggests an alternative two-stage iterated optimization algorithm: If we know the *expected* value of S , one could use this expectation in the first step to perform a weighted maximum likelihood estimation of Eq. 2.40 with respect to the model parameters. These estimates will be incorrect since the expectation S is inaccurate. So, in the second step, one could update the expected value of all S pretending the model parameters $\nu := (\pi, \theta)$ are known and held fixed at their values from the past iteration. This is precisely the strategy of the expectation–maximization (EM) algorithm—a statistically self-consistent, iterative, algorithm for maximum likelihood estimation. With the context of mixture models, the EM algorithm is outlined as follows:

- **E-step:**

In this step, the parameters ν are held fixed at the old values, ν^{old} , obtained from the previous iteration (or at their initial settings during the algorithm’s initialization). Conditioned on the observations, the E-step then computes the probability density of the state variables $S_i, \forall i$ given the current model parameters and observation data, i.e.

$$p(s_i | x_i, \nu^{old}) \propto p(x_i | s_i; \theta) p(s_i; \pi^{old}). \quad (2.41)$$

In particular, we compute

$$\mathbb{P}((s_i)_k = 1 \mid x_i, v^{old}) = \frac{p(x_i \mid (s_i)_k = 1; \theta_k)\pi_k}{\sum_{\ell} p(x_i \mid (s_i)_{\ell} = 1; \theta_{\ell})\pi_{\ell}}. \quad (2.42)$$

The likelihood terms $p(x_i \mid (s_i)_k = 1; \theta_k)$ are evaluated using the observation densities defined for each of the states.

- **M-step:**

In this step, the hidden state probabilities are considered given and maximization is performed with respect to the parameters:

$$v^{new} = \arg \max_v \mathcal{L}(v). \quad (2.43)$$

This results in the following update equations for the parameters in the probability distributions:

$$\mu_k = \frac{1}{N} \frac{\sum_{i=1}^N (\gamma_i)_k x_i}{\sum_{i=1}^N (\gamma_i)_k} \quad (2.44)$$

$$\sigma_k^2 = \frac{1}{N} \frac{\sum_{i=1}^N (\gamma_i)_k (x_i - \mu_k)^2}{\sum_{i=1}^N (\gamma_i)_k}, \quad \forall k \in \{1, \dots, K\}, \quad (2.45)$$

where $(\gamma_i)_k := \mathbb{E}[(s_i)_k \mid x_i]$ are the *responsibilities*—conditional expectations which measure how strongly a data point, x_i , “belongs” to each component, k , of the mixture model.

The number of components needed to model the data depends on the data and can be determined by a Kolmogorov-Smirnov test or based on entropy criteria. Heavy tailed data required at least two light tailed components to compensate. More components require larger sample sizes to ensure adequate fitting. In the extreme case there may be insufficient data available to calibrate a given mixture model with a certain degree of accuracy. In summary, while GMMs are flexible they may not be the most appropriate model. If more is known about the data distribution, such as its behavior in the tails, incorporation of this knowledge can only help improve the model.

? Multiple Choice Question 3

Which of the following statements are true:

1. Mixture models assume that the data is multi-modal and drawn from a linear combination of uni-modal distributions.
2. The expectation–maximization (EM) algorithm is a type of iterative unsupervised learning algorithm which alternates between updating the probability density of the state variables, based on model parameters (E-step) and updating the parameters by maximum likelihood estimation (M-step).

3. The EM algorithm automatically determines the modality of the distribution and hence the number of components.
 4. A mixture model is only appropriate for use in finance if the modeler specifies which component is the most relevant for each observation.
-

9 Summary

Probabilistic modeling is an important class of models in financial data, which is often noisy and incomplete. Additionally much of finance rests on being able to make financial decisions under uncertainty, a task perfectly suited to probabilistic modeling. In this chapter we have identified and demonstrated how probabilistic modeling is used for financial modeling. In particular we have:

- Applied Bayesian inference to data using simple probabilistic models;
- Show how linear regression with probabilistic weights can be viewed as a simple probabilistic graphical model; and
- Developed more versatile representations of complex data with probabilistic graphical models such as mixture models and hidden Markov models.

10 Exercises

Exercise 2.1: Applied Bayes' Theorem

An accountant is 95 percent effective in detecting fraudulent accounting when it is, in fact, present. However, the audit also yields a “false positive” result for one percent of the non-fraudulent companies audited. If 0.1 percent of the companies are actually fraudulent, what is the probability that a company is fraudulent given that the audit revealed fraudulent accounting?

Exercise 2.2*: FX and Equity

A currency strategist has estimated that JPY will strengthen against USD with probability 60% if S&P 500 continues to rise. JPY will strengthen against USD with probability 95% if S&P 500 falls or stays flat. We are in an upward trending market at the moment, and we believe that the probability that S&P 500 will rise is 70%. We then learn that JPY has actually strengthened against USD. Taking this new information into account, what is the probability that S&P 500 will rise? Hint: Recall Bayes' rule: $P(A | B) = \frac{P(B | A)}{P(B)} P(A)$.

Exercise 2.3**: Bayesian Inference in Trading

Suppose there are n conditionally independent, but not identical, Bernoulli trials G_1, \dots, G_n generated from the map $P(G_i = 1 | X = x_i) = g_1(x_i | \theta)$ with $\theta \in [0, 1]$. Show that the likelihood of $G | X$ is given by

$$p(G | X, \theta) = \prod_{i=1}^n (g_1(x_i | \theta))^{G_i} \cdot (g_0(x_i | \theta))^{1-G_i} \quad (2.46)$$

and the log-likelihood of $G | X$ is given by

$$\ln p(G | X, \theta) = \sum_{i=1}^n G_i \ln(g_1(x_i | \theta)) + (1 - G_i) \ln(g_0(x_i | \theta)). \quad (2.47)$$

Using Bayes' rule, write the condition probability density function of θ (the "posterior") given the data (X, G) in terms of the above likelihood function.

From the previous example, suppose that $G = 1$ corresponds to JPY strengthening against the dollar and X are the S&P 500 daily returns and now

$$g_1(x | \theta) = \theta \mathbb{1}_{x>0} + (\theta + 35) \mathbb{1}_{x \leq 0}. \quad (2.48)$$

Starting with a neutral view on the parameter θ (i.e., $\theta \in [0, 1]$), learn the distribution of the parameter θ given that JPY strengthens against the dollar for two of the three days and S&P 500 is observed to rise for 3 consecutive days. Hint: You can use the Beta density function with a scaling constant $\Gamma(\alpha, \beta)$

$$p(\theta | \alpha, \beta) = \frac{(\alpha + \beta - 1)!}{(\alpha - 1)!(\beta - 1)!} \theta^{\alpha-1} (1 - \theta)^{\beta-1} = \Gamma(\alpha, \beta) \theta^{\alpha-1} (1 - \theta)^{\beta-1} \quad (2.49)$$

to evaluate the integral in the marginal density function.

If θ represents the currency analyst's opinion of JPY strengthening against the dollar, what is the probability that the model overestimates the analyst's estimate?

Exercise 2.4*: Bayesian Inference in Trading

Suppose that you observe the following daily sequence of directional changes in the JPY/USD exchange rate (U (up), D(down or stays flat)):

U, D, U, U, D

and the corresponding daily sequence of S&P 500 returns is

-0.05, 0.01, -0.01, -0.02, 0.03

You propose the following probability model to explain the behavior of JPY against USD given the directional changes in S&P 500 returns: Let G denote a Bernoulli R.V., where $G = 1$ corresponds to JPY strengthening against the dollar and r are the S&P 500 daily returns. All observations of G are conditionally independent (but *not* identical) so that the likelihood is

$$p(G | r, \theta) = \prod_{i=1}^n p(G = G_i | r = r_i, \theta)$$

where

$$p(G_i = 1 \mid r = r_i, \theta) = \begin{cases} \theta_u, & r_i > 0 \\ \theta_d, & r_i \leq 0. \end{cases}$$

Compute the full expression for the likelihood that the data was generated by this model.

Exercise 2.5: Model Comparison

Suppose you observe the following daily sequence of direction changes in the stock market (U (up), D(down)):

U, D, U, U, D, D, D, U, U, U, U, U, U, U, D, U, D, U, D,
U, D, D, D, D, U, U, D, U, U, U, D, U, D, D, D, U, U,
D, D, D, U, D, U, D, U, D

You compare two models for explaining its behavior. The first model, \mathcal{M}_1 , assumes that the probability of an upward movement is fixed to 0.5 and the data is i.i.d.

The second model, \mathcal{M}_2 , also assumes the data is i.i.d. but that the probability of an upward movement is set to an unknown $\theta \in \Theta = (0, 1)$ with a uniform prior on $\theta : p(\theta | \mathcal{M}_2) = 1$. For simplicity, we additionally choose a uniform model prior $p(\mathcal{M}_1) = p(\mathcal{M}_2)$.

Compute the model evidence for each model.

Compute the Bayes' factor and indicate which model should we prefer in light of this data?

Exercise 2.6: Bayesian Prediction and Updating

Using Bayesian prediction, predict the probability of an upward movement given the best model and data in Exercise 2.5.

Suppose now that you observe the following new daily sequence of direction changes in the stock market (U (up), D(down)):

D, U, D, D, D, U, D, U, D, D, D, U, D, U, D, D, D, D,
U, U, D, D, D, U, D, U, D, D, D, U, D, U, D, U, D, D, D,
D, U, U, D, U, D, U

Using the best model from Exercise 2.5, compute the new posterior distribution function based on the new data and the data in the previous question and predict the probability of an upward price movement given all data. State all modeling assumptions clearly.

Exercise 2.7: Logistic Regression Is Naive Bayes

Suppose that G and $X \in \{0, 1\}^p$ are Bernoulli random variables and the X_i s are mutually independent given G —that is, $\mathbb{P}[X \mid G] = \prod_{i=1}^p \mathbb{P}[X_i \mid G]$. Given a naive Bayes' classifier $\mathbb{P}[G \mid X]$, show that the following logistic regression model produces equivalent output if the weights are

$$w_0 = \log \frac{\mathbb{P}[G]}{\mathbb{P}[G^c]} + \sum_{i=1}^p \log \frac{\mathbb{P}[X_i = 0 \in G]}{\mathbb{P}[X_i = 0 \in G^c]}$$

$$w_i = \log \frac{\mathbb{P}[X_i = 1 \in G]}{\mathbb{P}[X_i = 1 \in G^c]} \cdot \frac{\mathbb{P}[X_i = 0 \in G^c]}{\mathbb{P}[X_i = 0 \in G]}, \quad i = 1, \dots, p.$$

Exercise 2.8*: Restricted Boltzmann Machines**

Consider a probabilistic model with two types of binary variables: visible binary stochastic units $\mathbf{v} \in \{0, 1\}^D$ and hidden binary stochastic units $\mathbf{h} \in \{0, 1\}^F$, where D and F are the number of visible and hidden units, respectively. The joint probability density to observe their values is given by the exponential distribution

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h})), \quad Z = \sum_{\mathbf{v}, \mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$$

and where the energy $E(\mathbf{v}, \mathbf{h})$ of the state $\{\mathbf{v}, \mathbf{h}\}$ is

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T W \mathbf{h} - \mathbf{b}^T \mathbf{v} - \mathbf{a}^T \mathbf{h} = -\sum_{i=1}^D \sum_{j=1}^F W_{ij} v_i h_j - \sum_{i=1}^D b_i v_i - \sum_{j=1}^F a_j h_j,$$

with model parameters \mathbf{a} , \mathbf{b} , W . This probabilistic model is called the restricted Boltzmann machine. Show that conditional probabilities for visible and hidden nodes are given by the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$:

$$\mathbb{P}[v_i = 1 | \mathbf{h}] = \sigma \left(\sum_j W_{ij} h_j + b_i \right), \quad \mathbb{P}[h_i = 1 | \mathbf{v}] = \sigma \left(\sum_j W_{ij} v_j + a_i \right).$$

Appendix

Answers to Multiple Choice Questions

Question 1

Answer: 1,3,4.

Question 2

Answer: 1,3,5.

Question 3

Answer: 1,2. Mixture models assume that the data is multi-modal—the data is drawn from a linear combination of uni-modal distributions. The expectation–maximization (EM) algorithm is a type of iterative, self-consistent, unsupervised

learning algorithm which alternates between updating the probability density of the state variables, based on model parameters (E-step) and updating the parameters by maximum likelihood estimation (M-step). The EM algorithm does not automatically determine the modality of the data distribution, although there are statistical tests to determine this. A mixture model assigns a probabilistic weight for every component that each observation might belong to. The component with the highest weight is chosen.

References

- Breiman, L. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science* 16(3), 199–231.
- Duembgen, M., & Rogers, L. C. G. (2014). Estimate nothing. <https://arxiv.org/abs/1401.5666>.
- Finger, C. (1997). *A methodology to stress correlations*, fourth Quarter. RiskMetrics Monitor.
- Rasmussen, C. E., & Ghahramani, Z. (2001). Occam's razor. In *In Advances in Neural Information Processing Systems 13*, (pp. 294–300). MIT Press.

Chapter 3

Bayesian Regression and Gaussian Processes



This chapter introduces Bayesian regression and shows how it extends many of the concepts in the previous chapter. We develop kernel based machine learning methods—specifically Gaussian process regression, an important class of Bayesian machine learning methods—and demonstrate their application to “surrogate” models of derivative prices. This chapter also provides a natural starting point from which to develop intuition for the role and functional form of regularization in a frequentist setting—the subject of subsequent chapters.

1 Introduction

In general, it is difficult to develop intuition about how the distribution of weights in a parametric regression model represents the data. Rather than induce distributions over variables, as we have seen in the previous chapter, we could instead induce distributions over functions. Specifically, we can express those intuitions using a “covariance kernel.”

We start by exploring Bayesian regression in a more general setup that enables us to easily move from a toy regression model to a more complex non-parametric Bayesian regression model, such as Gaussian process regression. By introducing Bayesian regression in more depth, we show how it extends many of the concepts in the previous chapter. We develop kernel based machine learning methods (specifically Gaussian process regression), and demonstrate their application to “surrogate” models of derivative prices.¹

¹Surrogate models learn the output of an existing mathematical or statistical model as a function of input data.

Chapter Objectives

The key learning points of this chapter are:

- Formulate a Bayesian linear regression model;
 - Derive the posterior distribution and the predictive distribution;
 - Describe the role of the prior as an equivalent form of regularization in maximum likelihood estimation; and
 - Formulate and implement Gaussian Processes for kernel based probabilistic modeling, with programming examples involving derivative modeling.
-

2 Bayesian Inference with Linear Regression

Consider the following linear regression model which is affine in $x \in \mathbb{R}$:

$$y = f(x) = \theta_0 + \theta_1 x, \quad \theta_0, \theta_1 \sim \mathcal{N}(0, 1), \quad x \in \mathbb{R}, \quad (3.1)$$

and suppose that we observe the value of the function over the inputs $\mathbf{x} := [x_1, \dots, x_n]$. The random parameter vector $\boldsymbol{\theta} := [\theta_0, \theta_1]$ is unknown. This setup is referred to as “noise-free,” since we assume that \mathbf{y} is strictly given by the function $f(x)$ without noise.

The graphical model representation of this model is given in Fig. 3.1 and clearly specifies that the i th model output only depends on x_i . Note that the graphical model also holds in the case when there is noise.

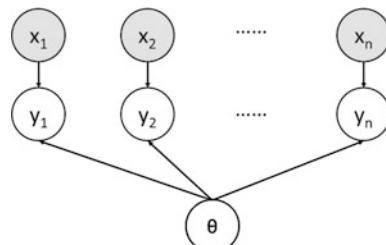
In the noise-free setting, the expectation of the function under known data is

$$\mathbb{E}_{\boldsymbol{\theta}}[f(x_i)|x_i] = \mathbb{E}_{\boldsymbol{\theta}}[\theta_0] + \mathbb{E}_{\boldsymbol{\theta}}[\theta_1]x_i = 0, \forall i, \quad (3.2)$$

where the expectation operator is w.r.t. the prior density of $\boldsymbol{\theta}$

$$\mathbb{E}_{\boldsymbol{\theta}}[\cdot] = \int (\cdot) p(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (3.3)$$

Fig. 3.1 This graphical model represents Bayesian linear regression. The features $\mathbf{x} := \{x_i\}_{i=1}^n$ and responses $\mathbf{y} := \{y_i\}_{i=1}^n$ are known and the random parameter vector $\boldsymbol{\theta}$ is unknown. The i th model output only depends on x_i



Then the covariance of the function values between any two points, x_i and x_j is

$$\mathbb{E}_{\theta}[f(x_i)f(x_j)|x_i, x_j] = \mathbb{E}_{\theta}[\theta_0^2 + \theta_0\theta_1(x_i + x_j) + \theta_1^2x_i x_j] \quad (3.4)$$

$$= \mathbb{E}_{\theta}[\theta_0^2] + \mathbb{E}_{\theta}[\theta_0^2]x_i x_j + \mathbb{E}_{\theta}[\theta_0\theta_1](x_i + x_j), \quad (3.5)$$

$$= 1 + x_i x_j, \quad (3.6)$$

where the last term is zero because of the independence of θ_0 and θ_1 . Then any collection of function values $[f(x_1), \dots, f(x_n)]$ with given data has a joint Gaussian distribution with covariance matrix $K_{ij} := \mathbb{E}_{\theta}[f(x_i)f(x_j)|x_i, x_j] = 1 + x_i x_j$. Such a probabilistic model is the simplest example of a more general, non-linear, Bayesian kernel learning method referred to as “Gaussian Process Regression” or simply “Gaussian Processes” (GPs) and is the subject of the later material in this chapter.

Noisy Data

The above example is in a noise-free setting where the function values $[f(x_1), \dots, f(x_n)]$ are observed. In practice, we do not observe these function values, but rather some target values $\mathbf{y} = [y_1, \dots, y_n]$ which depend on \mathbf{x} by the function, $f(\mathbf{x})$, and some zero-mean Gaussian i.i.d. additive noise with known variance σ_n^2 :

$$y_i = f(x_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_n^2). \quad (3.7)$$

Hence the observed i.i.d. data is $\mathcal{D} := (\mathbf{x}, \mathbf{y})$. Following Rasmussen and Williams (2006), under this noise assumption and the linear model we can write down the likelihood function of the data:

$$\begin{aligned} p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) &= \prod_{i=1}^n p(y_i|x_i, \boldsymbol{\theta}) \\ &= \frac{1}{\sqrt{2\pi}\sigma_n} \exp\{-(y_i - x_i\theta_1 - \theta_0)^2/(2\sigma_n^2)\} \end{aligned}$$

and hence $\mathbf{y}|\mathbf{x}, \boldsymbol{\theta} \sim \mathcal{N}(\theta_0 + \theta_1\mathbf{x}, \sigma_n^2 I)$.

Bayesian inference of the parameters in this linear regression model is based on the posterior distribution over the weights:

$$p(\theta_i|\mathbf{y}, \mathbf{x}) = \frac{p(\mathbf{y}|\mathbf{x}, \theta_i)p(\theta_i)}{p(\mathbf{y}|\mathbf{x})}, \quad i \in \{0, 1\}, \quad (3.8)$$

where the marginal likelihood in the denominator is given by integrating over the parameters as

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (3.9)$$

If we define the matrix X , where $[X]_i := [1, x_i]$, and under more general conjugate priors, we have

$$\begin{aligned}\boldsymbol{\theta} &\sim \mathcal{N}(\boldsymbol{\mu}, \Sigma), \\ \mathbf{y}|X, \boldsymbol{\theta} &\sim \mathcal{N}(\boldsymbol{\theta}^T X, \sigma_n^2 I),\end{aligned}$$

and the product of Gaussian densities is also Gaussian, we can simply use standard results of moments of affine transformations to give

$$\mathbb{E}[\mathbf{y}|X] = \mathbb{E}[\boldsymbol{\theta}^T X + \epsilon] = \mathbb{E}[\boldsymbol{\theta}^T]X = \boldsymbol{\mu}^T X. \quad (3.10)$$

The conditional covariance is

$$Cov(\mathbf{y}|X) = Cov(\boldsymbol{\theta}^T X) + \sigma_n^2 I = XCov(\boldsymbol{\theta})X^T + \sigma_n^2 I = X\Sigma X^T + \sigma_n^2 I. \quad (3.11)$$

To derive the posterior of $\boldsymbol{\theta}$, it is convenient to transform the prior density function from a moment parameterization to a natural parameterization by completing the square. This is useful for multiplying normal density functions such as normalized likelihoods and conjugate priors. The quadratic form for the prior transforms to

$$p(\boldsymbol{\theta}) \propto \exp\left\{-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu})\right\}, \quad (3.12)$$

$$\propto \exp\left\{\boldsymbol{\mu}^T \Sigma^{-1} \boldsymbol{\theta} - \frac{1}{2} \boldsymbol{\theta}^T \Sigma^{-1} \boldsymbol{\theta}\right\}, \quad (3.13)$$

where the $\frac{1}{2}\boldsymbol{\mu}^T \Sigma^{-1} \boldsymbol{\mu}$ term is absorbed in the normalizing term as it is independent of $\boldsymbol{\theta}$. Using this transformation, the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ is proportional to:

$$p(\mathbf{y}|X, \boldsymbol{\theta})p(\boldsymbol{\theta}) \propto \exp\left\{-\frac{1}{2\sigma_n^2}(\mathbf{y} - \boldsymbol{\theta}^T X)^T(\mathbf{y} - \boldsymbol{\theta}^T X)\right\} \exp\left\{\boldsymbol{\mu}^T \Sigma^{-1} - \frac{1}{2} \boldsymbol{\theta}^T \Sigma^{-1} \boldsymbol{\theta}\right\} \quad (3.14)$$

$$\propto \exp\left\{-\frac{1}{2\sigma_n^2}(-2\mathbf{y}\boldsymbol{\theta}^T X + \boldsymbol{\theta}^T X X^T \boldsymbol{\theta})\right\} \exp\left\{\boldsymbol{\mu}^T \Sigma^{-1} - \frac{1}{2} \boldsymbol{\theta}^T \Sigma^{-1} \boldsymbol{\theta}\right\} \quad (3.15)$$

$$= \exp\left\{(\Sigma^{-1}\boldsymbol{\mu} + \frac{1}{\sigma_n^2}\mathbf{y}^T X)^T \boldsymbol{\theta}^T - \frac{1}{2} \boldsymbol{\theta}^T (\Sigma^{-1} + \frac{1}{\sigma_n^2} X X^T) \boldsymbol{\theta}\right\} \quad (3.16)$$

$$= \exp\left\{a^T \boldsymbol{\theta} - \frac{1}{2} \boldsymbol{\theta}^T A \boldsymbol{\theta}\right\}. \quad (3.17)$$

The posterior follows the distribution

$$\boldsymbol{\theta} | \mathcal{D} \sim \mathcal{N}(\mu', \Sigma'), \quad (3.18)$$

where the moments of the posterior are

$$\mu' = \Sigma' \boldsymbol{a} = (\Sigma^{-1} + \frac{1}{\sigma_n^2} \mathbf{X} \mathbf{X}^T)^{-1} (\Sigma^{-1} \boldsymbol{\mu} + \frac{1}{\sigma_n^2} \mathbf{y}^T \mathbf{X}) \quad (3.19)$$

$$\Sigma' = A^{-1} = (\Sigma^{-1} + \frac{1}{\sigma_n^2} \mathbf{X} \mathbf{X}^T)^{-1} \quad (3.20)$$

and we use the inverse of transformation above, from natural back to moment parameterization to write

$$p(\boldsymbol{\theta} | \mathcal{D}) \propto \exp\left\{-\frac{1}{2}(\boldsymbol{\theta} - \mu')^T (\Sigma')^{-1} (\boldsymbol{\theta} - \mu')\right\}. \quad (3.21)$$

Σ^{-1} , the inverse of a covariance matrix, is referred to as the *precision matrix*. The mean of this distribution is the maximum a posteriori (MAP) estimate of the weights—it is the mode of the posterior distribution. We will show shortly that it corresponds to the penalized maximum likelihood estimate of the weights, with a L_2 (ridge) penalty term given by the log prior.

Figure 3.2 demonstrates Bayesian learning of the posterior distribution of the weights. A bi-variate Gaussian prior is initially chosen for the prior distribution and there are an infinite number of possible lines that could be drawn in the data space $[-1, 1] \times [-1, 1]$. The data is generated under the model $f(x) = 0.3 + 0.5x$ with a small amount of additive i.i.d. Gaussian noise. As the number of points that the likelihood function is evaluated over increases, the posterior distribution sharpens and eventually contracts to a point. See the Bayesian Linear regression Python notebook for details of the implementation.

? Multiple Choice Question 1

Which of the following statements are true:

1. Bayesian regression treats the regression weights as random variables.
2. In Bayesian regression the data function $f(x)$ is assumed to always be observed.
3. The posterior distribution of the parameters is always Gaussian if the prior is Gaussian.
4. The posterior distribution of the regression weights will typically contract with increasing data.
5. The mean of the posterior distribution depends on both the mean and covariance of the prior if it is Gaussian.

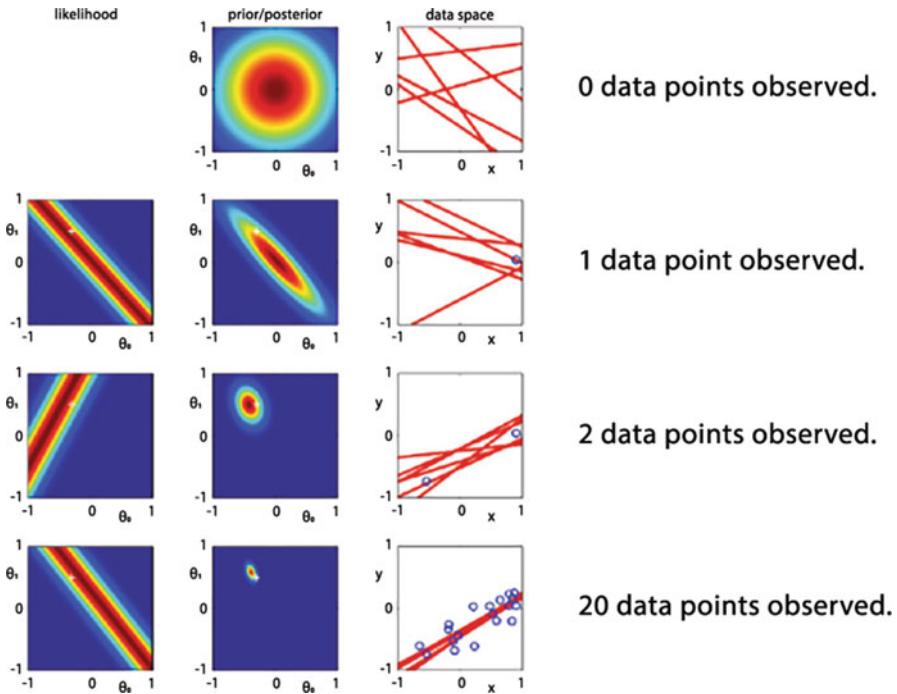


Fig. 3.2 This figure demonstrates Bayesian inference for the linear model. The data has been generated from the function $f(x) = 0.3 + 0.5x$ with a small amount of additive white noise. Source: Bishop (2006)

2.1 Maximum Likelihood Estimation

Let us briefly revisit parameter estimation in a frequentist setting to solidify our understanding of Bayesian inference. Assuming that σ_n^2 is a known parameter, we can easily derive the maximum likelihood estimate of the parameter vector, $\hat{\theta}$. The gradient of the negative log-likelihood function (a.k.a. loss function) w.r.t. θ is

$$\begin{aligned} \frac{d}{d\theta} \mathcal{L}(\theta) &:= -\frac{d}{d\theta} \left(\sum_{i=1}^n \log p(y_i | x_i, \theta) \right), \\ &= \frac{1}{2\sigma_n^2} \frac{d}{d\theta} \left(\|\mathbf{y} - \theta^T \mathbf{X}\|_2^2 + c \right) \\ &= \frac{1}{\sigma_n^2} (-\mathbf{y}^T \mathbf{X} + \theta^T \mathbf{X}^T \mathbf{X}), \end{aligned}$$

where the constant $c := -\frac{n}{2}(\log(2\pi) + \log(\sigma_n^2))$. Setting this gradient to zero gives the orthogonal projection of \mathbf{y} on to the subspace spanned by \mathbf{X} :

$$\hat{\boldsymbol{\theta}} = (X^T X)^{-1} X^T \mathbf{y}, \quad (3.22)$$

where $\hat{\boldsymbol{\theta}}$ is the vector in the subspace spanned by X which is closest to \mathbf{y} . This result states that the maximum likelihood estimate of an unpenalized loss function (i.e., without including the prior) is the OLS estimate when the noise variance is known. If the noise variance is unknown then the loss function is

$$\mathcal{L}(\boldsymbol{\theta}, \sigma_n^2) = \frac{n}{2} \log(\sigma_n^2) + \frac{1}{2\sigma_n^2} \|\mathbf{y} - \boldsymbol{\theta}^T X\|_2^2 + c, \quad (3.23)$$

where now $c = \frac{n}{2} \log(2\pi)$. Taking the partial derivative

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \sigma_n^2)}{\partial \sigma_n^2} = \frac{n}{2\sigma_n^2} - \frac{1}{2\sigma_n^4} \|\mathbf{y} - \boldsymbol{\theta}^T X\|_2^2, \quad (3.24)$$

and setting it to zero gives ${}^2 \hat{\sigma}_n^2 = \frac{1}{n} \|\mathbf{y} - \boldsymbol{\theta}^T X\|_2^2$.

Maximum likelihood estimation is prone to overfitting and therefore should be avoided. We instead maximize the posterior distribution to arrive at the MAP estimate, $\hat{\boldsymbol{\theta}}_{MAP}$. Returning to the above computation under known noise:

$$\begin{aligned} \frac{d}{d\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) &:= -\frac{d}{d\boldsymbol{\theta}} \left(\sum_{i=1}^n \log p(y_i | x_i, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \right), \\ &= \frac{d}{d\boldsymbol{\theta}} \left(\frac{1}{2\sigma_n^2} \|\mathbf{y} - \boldsymbol{\theta}^T X\|_2^2 + \frac{1}{2} (\boldsymbol{\theta} - \mu)^T \Sigma^{-1} (\boldsymbol{\theta} - \mu) + c \right) \\ &= \frac{1}{\sigma_n^2} (-\mathbf{y}^T X + \boldsymbol{\theta}^T X X^T) + (\boldsymbol{\theta} - \mu)^T \Sigma^{-1}. \end{aligned}$$

Setting this derivative to zero gives

$$\frac{1}{\sigma_n^2} (\mathbf{y}^T X - \boldsymbol{\theta}^T X X^T) = (\boldsymbol{\theta} - \mu)^T \Sigma^{-1}, \quad (3.25)$$

and after some rearrangement we obtain

$$\hat{\boldsymbol{\theta}}_{MAP} = (X X^T + \sigma_n^2 \Sigma^{-1})^{-1} (\sigma_n^2 \Sigma^{-1} \mu + X^T \mathbf{y}) = A^{-1} (\Sigma^{-1} \mu + \sigma_n^{-2} X^T \mathbf{y}), \quad (3.26)$$

which is equal to the mean of the posterior derived in Eq. 3.19. Of course, this is to be expected since the mean of a Gaussian distribution is also its mode. The difference between $\hat{\boldsymbol{\theta}}_{MAP}$ and $\hat{\boldsymbol{\theta}}$ are the $\sigma_n^2 \Sigma^{-1}$ terms. This term has the effect of

²Note that the factor of 2 in the denominator of the second term does not cancel out because the derivative is w.r.t. σ_n^2 and not σ_n .

reducing the condition number of $X^T X$. Forgetting the mean of the prior, the linear system $(X^T X)\theta = X^T \mathbf{y}$ becomes the regularized linear system: $A\theta = \sigma_n^{-2} X^T \mathbf{y}$.

Note that choosing the isotropic Gaussian prior $p(\theta) = \mathcal{N}(0, \frac{1}{2\lambda} I)$ gives the ridge penalty term in the loss function: $\lambda ||\theta||_2^2$, i.e. the negative log Gaussian prior matches the ridge penalty term up to a constant. In the limit, $\lambda \rightarrow 0$ recovers maximum likelihood estimation—this corresponds to using the uninformative prior.

Of course, in Bayesian inference, we do not perform point-estimation of the parameters, however it was a useful exercise to confirm that the mean of the posterior in Eq. 3.19 did indeed match the MAP estimate. Furthermore, we have made explicit the interpretation of the prior as a regularization term used in ridge regression.

2.2 Bayesian Prediction

Recall from Chap. 2 that Bayesian prediction requires evaluating the density of $f_* := f(x_*)$ w.r.t. a new data point x_* and the training data \mathcal{D} .

In general, we predict the model output at a new point, f_* , by averaging the model output over all possible weights, with the weight density function given by the posterior. That is we seek to find the marginal density $p(f_*|x_*, \mathcal{D}) = \mathbb{E}_{\theta|\mathcal{D}}[p(f_*|x_*, \theta)]$, where the dependency on θ has been integrated out. This conditional density is Gaussian

$$f_*|x_*, \mathcal{D} \sim \mathcal{N}(\mu_*, \Sigma_*), \quad (3.27)$$

with moments

$$\begin{aligned} \mu_* &= \mathbb{E}_{\theta|\mathcal{D}}[f_*|x_*, \mathcal{D}] = x_*^T \mathbb{E}_{\theta|\mathcal{D}}[\theta|x_*, \mathcal{D}] = x_*^T \mathbb{E}_{\theta|\mathcal{D}}[\theta|\mathcal{D}] = x_*^T \mu' \\ \Sigma_* &= \mathbb{E}_{\theta|\mathcal{D}}[(f_* - \mu_*)(f_* - \mu_*)^T | x_*, \mathcal{D}] \\ &= x_*^T \mathbb{E}_{\theta|\mathcal{D}}[(\theta - \mu')(\theta - \mu')^T | x_*, \mathcal{D}] x_* \\ &= x_*^T \mathbb{E}_{\theta|\mathcal{D}}[(\theta - \mu')(\theta - \mu')^T |\mathcal{D}] x_* = x_*^T \Sigma' x_*, \end{aligned}$$

where we have avoided taking the expectation of the entire density function $p(f_*|x_*, \theta)$, but rather just the moments because we know that f_* is Gaussian.

? Multiple Choice Question 2

Which of the following statements are true:

1. Prediction under a Bayesian linear model requires first estimating the posterior distribution of the parameters;
2. The predictive distribution is Gaussian only if the posterior and likelihood distributions are Gaussian;

3. The predictive distribution depends on the weights in the models;
 4. The variance of the predictive distribution typically contracts with increasing training data.
-

2.3 Schur Identity

There is another approach to deriving the predictive distribution from the conditional distribution of the model output which relies on properties of inverse matrices. We can write the joint density between Gaussian random variables X and Y in terms of the *partitioned covariance matrix*:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \mathcal{N} \left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix} \right),$$

where $\Sigma_{xx} = \mathbb{V}(X)$, $\Sigma_{xy} = \text{Cov}(XY)$ and $\Sigma_{yy} = \mathbb{V}(Y)$, how can we find the conditional density $p(y|x)$?

In order to express the moments in terms of the partitioned covariance matrix we shall use the following Schur identity:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{bmatrix}.$$

where the Schur complement w.r.t. the submatrix D is $M := (A - BD^{-1}C)^{-1}$. Applying the Schur identity to the partitioned precision matrix A gives

$$\begin{bmatrix} \Sigma_{yy} & \Sigma_{yx} \\ \Sigma_{xy} & \Sigma_{xx} \end{bmatrix}^{-1} = \begin{bmatrix} A_{yy} & A_{yx} \\ A_{xy} & A_{xx} \end{bmatrix}, \quad (3.28)$$

where

$$A_{yy} = (\Sigma_{yy} - \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy})^{-1} \quad (3.29)$$

$$A_{yx} = -(\Sigma_{yy} - \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy})^{-1}\Sigma_{yx}\Sigma_{xx}^{-1}, \quad (3.30)$$

and thus the moments of the Gaussian distribution $p(y|x)$ are

$$\mu_{y|x} = \mu_y + \Sigma_{yx}\Sigma_{xx}^{-1}(x - \mu_x), \quad (3.31)$$

$$\Sigma_{y|x} = \Sigma_{yy} - \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy}. \quad (3.32)$$

Hence the density of the condition distribution $Y|X$ can alternatively be derived by using the Schur identity. In the special case when the joint density $p(x, y)$ is bi-Gaussian, the expression for the moments simplify to

$$\mu_{y|x} = \mu_y + \frac{\sigma_{yx}}{\sigma_x^2}(x - \mu_x), \quad (3.33)$$

$$\Sigma_{y|x} = \sigma_y - \frac{\sigma_{yx}^2}{\sigma_x^2}, \quad (3.34)$$

where σ_{xy} is the covariance between X and Y .

Now returning to the predictive distribution, the joint density between \mathbf{y} and f_* is

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} = \mathcal{N} \left(\begin{bmatrix} \mu_y \\ \mu_{f_*} \end{bmatrix}, \begin{bmatrix} \Sigma_{yy} & \Sigma_{yf_*} \\ \Sigma_{f_*y} & \Sigma_{f_*f_*} \end{bmatrix} \right). \quad (3.35)$$

We can immediately write down the moments of the condition distribution

$$\mu_{f_*|X,y,x_*} = \mu_{f_*} + \Sigma_{f_*y} \Sigma_{yy}^{-1} (y - \mu_y), \quad (3.36)$$

$$\Sigma_{f_*|X,y,x_*} = \Sigma_{f_*f_*} - \Sigma_{f_*y} \Sigma_{yy}^{-1} \Sigma_{yf_*}. \quad (3.37)$$

Since we know the form of the function $f(x)$, we can simplify this expression by writing that

$$\Sigma_{yy} = K_{X,X} + \sigma_n^2 I, \quad (3.38)$$

where $K_{X,X}$ is the covariance of $f(X)$, which for linear regression takes the form

$$K_{X,X} = \mathbb{E}[\theta_1^2 (X - \mu_x)^2],$$

$\Sigma_{f_*f_*} = K_{x_*,x_*}$ and $\Sigma_{yf_*} = K_{X,x_*}$. Now we can write the moments of the predictive distribution as

$$\mu_{f_*|X,y,x_*} = \mu_{f_*} + K_{x_*,X} K_{X,X}^{-1} (y - \mu_y), \quad (3.39)$$

$$K_{f_*|X,y,x_*} = K_{x_*,x_*} - K_{x_*,X} K_{X,X}^{-1} K_{X,x_*}. \quad (3.40)$$

Discussion

Note that we have assumed that the functional form of the map, $f(x)$ is known and parameterized. Here we assumed that the map is linear in the parameters and affine in the features. Hence our approximation of the map is in the data space and, for prediction, we can subsequently forget about the map and work with its moments. The moments of the prior on the weights also no longer need to be specified.

If we do not know the form of the map but want to specify structure on the covariance of the map (i.e., the kernel), then we are said to be approximating in the kernel space rather than in the data space. If the kernels are given by continuous functions of X , then such an approximation corresponds to learning a posterior distribution over an infinite dimensional function space rather than a finite dimensional vector space. Put differently, we perform non-parametric regression rather than parametric regression. This is the remaining topic of this chapter and is precisely how Gaussian process regression models data.

3 Gaussian Process Regression

Whereas, statistical inference involves learning a latent function $Y = f(X)$ of the training data, $(X, Y) := \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \dots, n\}$, the idea of GPs is to, without parameterizing³ $f(X)$, place a probabilistic prior directly on the space of functions (MacKay 1998). Restated, the GP is hence a Bayesian non-parametric model that generalizes the Gaussian distributions from finite dimensional vector spaces to infinite dimensional function spaces. GPs do not provide some parameterized map, $\hat{Y} = f_\theta(X)$, but rather the posterior distribution of the latent function given the training data.

The basic theory of prediction with GPs dates back to at least as far as the time series work of Kolmogorov or Wiener in the 1940s (see (Whittle and Sargent 1983)). GPs are an example of a more general class of supervised machine learning techniques referred to as “kernel learning,” which model the covariance matrix from a set of parameterized kernels over the input. GPs extend and put in a Bayesian framework spline or kernel interpolators, and Tikhonov regularization (see (Rasmussen and Williams 2006) and (Alvarez et al. 2012)). On the other hand, (Neal 1996) observed that certain neural networks with one hidden layer converge to a Gaussian process in the limit of an infinite number of hidden units.

We refer to the reader to (Rasmussen and Williams 2006) for an excellent introduction to GPs. In addition to a number of favorable statistical and mathematical properties, such as universality (Micchelli et al. 2006), the implementation support infrastructure is mature—provided by GpyTorch, scikit-learn, Edward, STAN, and other open-source machine learning packages.

In this section we restrict ourselves to the simpler case of single-output GPs where f is real-valued. Multi-output GPs are considered in the next section.

³This is in contrast to non-linear regressions commonly used in finance, which attempt to parameterize a non-linear function with a set of weights.

3.1 Gaussian Processes in Finance

The adoption of GPs in financial derivative modeling is more recent and sometimes under the name of “kriging” (see, e.g., (Cousin et al. 2016) or (Ludkovski 2018)). Examples of applying GPs to financial time series prediction are presented in (Roberts et al. 2013). These authors helpfully note that AR(p) processes are discrete-time equivalents of GP models with a certain class of covariance functions, known as Matérn covariance functions. Hence, GPs can be viewed as a Bayesian non-parametric generalization of well-known econometrics techniques. da Barrosa et al. (2016) present a GP method for optimizing financial asset portfolios. Other examples of GPs include metamodeling for expected shortfall computations (Liu and Staum 2010), where GPs are used to infer portfolio values in a scenario based on inner-level simulation of nearby scenarios, and Crépey and Dixon (2020), where multiple GPs infer derivative prices in a portfolio for market and credit risk modeling. The approach of Liu and Staum (2010) significantly reduces the required computational effort by avoiding inner-level simulation in every scenario and naturally takes account of the variance that arises from inner-level simulation. The caveat is that the portfolio remains fixed. The approach of Crépey and Dixon (2020), on the other hand, allows for the composition of the portfolio to be changed, which is especially useful for portfolio sensitivity analysis, risk attribution and stress testing.

Derivative Pricing, Greeking, and Hedging

In the general context of derivative pricing, Spiegeleer et al. (2018) noted that many of the calculations required for pricing a wide array of complex instruments, are often similar. The market conditions affecting OTC derivatives may often only slightly vary between observations by a few variables, such as interest rates. Accordingly, for fast derivative pricing, greeking, and hedging, Spiegeleer et al. (2018) propose offline learning the pricing function, through Gaussian Process regression. Specifically, the authors configure the training set over a grid and then use the GP to interpolate at the test points. We emphasize that such GP estimates depend on option pricing models, rather than just market data - somewhat counter the motivation for adopting machine learning, but also the case in other computational finance applications such as Hernandez (2017), Weinan et al. (2017), or Hans Bühler et al. (2018).

Spiegeleer et al. (2018) demonstrate the speed up of GPs relative to Monte-Carlo methods and tolerable accuracy loss applied to pricing and Greek estimation with a Heston model, in addition to approximating the implied volatility surface. The increased expressibility of GPs compared to cubic spline interpolation, a popular numerical approximation techniques useful for fast point estimation, is also demonstrated. However, the applications shown in (Spiegeleer et al. 2018) are limited to single instrument pricing and do not consider risk modeling aspects. In particular, their study is limited to single-output GPs, without consideration

of multi-output GPs (respectively referred to as single- vs. multi-GPs for brevity hereafter).

By contrast, multi-GPs directly model the uncertainty in the prediction of a vector of derivative prices (responses) with spatial covariance matrices specified by kernel functions. Thus the amount of error in a portfolio value prediction, at any point in space and time, can only be adequately modeled using multi-GPs (which, however, do not provide any methodology improvement in estimation of the mean with respect to single-GPs). See Crépey and Dixon (2020) for further details of how multi-GPs can be applied to estimate market and credit risk.

The need for uncertainty quantification in the prediction is certainly a practical motivation for using GPs, as opposed to frequentist machine learning techniques such as neural networks, etc., which only provide point estimates. A high uncertainty in a prediction might result in a GP model estimate being rejected in favor of either retraining the model or even using full derivative model repricing. Another motivation for using GPs, as we will see, is the availability of a scalable training method for the model hyperparameters.

3.2 Gaussian Processes Regression and Prediction

We say that a random function $f : \mathbb{R}^p \mapsto \mathbb{R}$ is drawn from a GP with a mean function μ and a covariance function, called kernel, k , i.e. $f \sim \mathcal{GP}(\mu, k)$, if for any input points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ in \mathbb{R}^p , the corresponding vector of function values is Gaussian:

$$[f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_n)] \sim \mathcal{N}(\boldsymbol{\mu}, K_{X,X}),$$

for some mean vector $\boldsymbol{\mu}$, such that $\boldsymbol{\mu}_i = \mu(\mathbf{x}_i)$, and covariance matrix $K_{X,X}$ that satisfies $(K_{X,X})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. We follow the convention⁴ in the literature of assuming $\boldsymbol{\mu} = \mathbf{0}$.

Kernels k can be any symmetric positive semidefinite function, which is the infinite dimensional analogue of the notion of a symmetric positive semidefinite (i.e., covariance) matrix, i.e. such that

$$\sum_{i,j=1}^n k(\mathbf{x}_i, \mathbf{x}_j) \xi_i \xi_j \geq 0, \text{ for any points } \mathbf{x}_k \in \mathbb{R}^p \text{ and reals } \xi_k.$$

Radial basis functions (RBF) are kernels that only depend on $||\mathbf{x} - \mathbf{x}'||$, such as the squared exponential (SE) kernel

⁴This choice is not a real limitation in practice (since it is for the prior) and does not prevent the mean of the predictor from being nonzero.

$$k(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{1}{2\ell^2} \|\mathbf{x} - \mathbf{x}'\|^2\right\}, \quad (3.41)$$

where the length-scale parameter ℓ can be interpreted as “how far you need to move in input space for the function values to become uncorrelated,” or the Matern (MA) kernel

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{\|\mathbf{x} - \mathbf{x}'\|}{\ell} \right)^\nu K_\nu \left(\sqrt{2\nu} \frac{\|\mathbf{x} - \mathbf{x}'\|}{\ell} \right) \quad (3.42)$$

(which converges to (3.41) in the limit where ν goes to infinity), where Γ is the gamma function, K_ν is the modified Bessel function of the second kind, and ℓ and ν are non-negative parameters.

GPs can be seen as distributions over the reproducing kernel Hilbert space (RKHS) of functions which is uniquely defined by the kernel function, k (Scholkopf and Smola 2001). GPs with RBF kernels are known to be universal approximators with prior support to within an arbitrarily small epsilon band of any continuous function (Micchelli et al. 2006).

Assuming additive Gaussian noise, $y \mid \mathbf{x} \sim \mathcal{N}(f(\mathbf{x}), \sigma_n^2)$, and a GP prior on $f(\mathbf{x})$, given training inputs $\mathbf{x} \in X$ and training targets $\mathbf{y} \in Y$, the predictive distribution of the GP evaluated at an arbitrary test point $\mathbf{x}_* \in X_*$ is:

$$\mathbf{f}_* \mid X, Y, \mathbf{x}_* \sim \mathcal{N}(\mathbb{E}[\mathbf{f}_*|X, Y, \mathbf{x}_*], \text{var}[\mathbf{f}_*|X, Y, \mathbf{x}_*]), \quad (3.43)$$

where the moments of the posterior over X_* are

$$\begin{aligned} \mathbb{E}[\mathbf{f}_*|X, Y, X_*] &= \boldsymbol{\mu}_{X_*} + K_{X_*, X} [K_{X, X} + \sigma_n^2 I]^{-1} Y, \\ \text{var}[\mathbf{f}_*|X, Y, X_*] &= K_{X_*, X_*} - K_{X_*, X} [K_{X, X} + \sigma_n^2 I]^{-1} K_{X, X_*}. \end{aligned} \quad (3.44)$$

Here, $K_{X_*, X}$, K_{X, X_*} , $K_{X, X}$, and K_{X_*, X_*} are matrices that consist of the kernel, $k : \mathbb{R}^p \times \mathbb{R}^p \mapsto \mathbb{R}$, evaluated at the corresponding points, X and X_* , and $\boldsymbol{\mu}_{X_*}$ is the mean function evaluated on the test inputs X_* .

One key advantage of GPs over interpolation methods is their expressibility. In particular, one can combine the kernels, using convolution, to generalize the base kernels (c.f. “multi-kernel” GPs (Melkumyan and Ramos 2011)).

3.3 Hyperparameter Tuning

GPs are fit to the data by optimizing *the evidence*-the marginal probability of the data given the model with respect to the learned kernel hyperparameters.

The evidence has the form (see, e.g., (Murphy 2012, Section 15.2.4, p. 523)):

$$\log p(Y | X, \lambda) = - \left[Y^\top (K_{X,X} + \sigma_n^2 I)^{-1} Y + \log \det(K_{X,X} + \sigma_n^2 I) \right] - \frac{n}{2} \log 2\pi, \quad (3.45)$$

where $K_{X,X}$ implicitly depends on the kernel hyperparameters λ (e.g., $[\ell, \sigma]$, assuming an SE kernel as per (3.41) or an MA kernel for some exogenously fixed value of ν in (3.42)).

The first and second term in the $[\dots]$ in (3.45) can be interpreted as a *model fit* and a *complexity penalty* term (see (Rasmussen and Williams 2006, Section 5.4.1)). Maximizing the evidence with respect to the kernel hyperparameters, i.e. computing $\lambda^* = \operatorname{argmax}_\lambda \log p(\mathbf{y} | \mathbf{x}, \lambda)$, results in an automatic Occam's razor (see (Alvarez et al. 2012, Section 2.3) and (Rasmussen and Ghahramani 2001)), through which we effectively learn the structure of the space of functional relationships between the inputs and the targets. In practice, the negative evidence is minimized by stochastic gradient descent (SGD). The gradient of the evidence is given analytically by

$$\partial_\lambda \log p(\mathbf{y} | \mathbf{x}, \lambda) = \operatorname{tr} \left((\boldsymbol{\alpha} \boldsymbol{\alpha}^T - (K + \sigma_n^2 I)^{-1}) \partial_\lambda (K + \sigma_n^2 I)^{-1} \right), \quad (3.46)$$

where $\boldsymbol{\alpha} := (K + \sigma_n^2 I)^{-1} \mathbf{y}$ and

$$\partial_\ell (K + \sigma_n^2 I)^{-1} = -(K + \sigma_n^2 I)^{-2} \partial_\ell K, \quad (3.47)$$

$$\partial_\sigma (K + \sigma_n^2 I)^{-1} = -2\sigma (K + \sigma_n^2 I)^{-2}, \quad (3.48)$$

with

$$\partial_\ell k(\mathbf{x}, \mathbf{x}') = \ell^{-3} ||\mathbf{x} - \mathbf{x}'||^2 k(\mathbf{x}, \mathbf{x}'). \quad (3.49)$$

? Multiple Choice Question 3

Which of the following statements are true:

1. Gaussian Processes are a Bayesian modeling approach which assumes that the data is Gaussian distributed.
2. Gaussian Processes place a probabilistic prior directly on the space of functions.
3. Gaussian Processes model the posterior of the predictor using a parameterized kernel representation of the covariance matrix.
4. Gaussian Processes can be fitted to data by maximizing the evidence for the kernel parameters.
5. During evidence maximization, different kernels are evaluated, and the optimal kernel is chosen.

3.4 Computational Properties

If uniform grids are used (as opposed to a mesh-free GP as described in Sect. 5.2), we have $n = \prod_{k=1}^p n_k$, where n_k are the number of grid points per variable.

However, although each kernel matrix $K_{X,X}$ is $n \times n$, we only store the n-vector α in (3.46), which brings reduced memory requirements.

Training time, required for maximizing (3.45) numerically, scales poorly with the number of observations n . This complexity stems from the need to solve linear systems and compute log determinants involving an $n \times n$ symmetric positive definite covariance matrix K . This task is commonly performed by computing the Cholesky decomposition of K incurring $O(n^3)$ complexity. Prediction, however, is faster and can be performed in $O(n^2)$ with a matrix–vector multiplication for each test point, and hence the primary motivation for using GPs is real-time risk estimation performance.

Online Learning

If the option pricing model is recalibrated intra-day, then the corresponding GP model should be retrained. Online learning techniques permit performing this incrementally (Pillonetto et al. 2010). To enable online learning, the training data should be augmented with the constant model parameters. Each time the parameters are updated, a new observation $(\mathbf{x}', \mathbf{y}')$ is generated from the option model prices under the new parameterization. The posterior at test point \mathbf{x}_* is then updated with the new training point following

$$p(\mathbf{f}_*|X, Y, \mathbf{x}', \mathbf{y}', \mathbf{x}_*) = \frac{p(\mathbf{x}', \mathbf{y}'|\mathbf{f}_*)p(\mathbf{f}_*|X, Y, \mathbf{x}_*)}{\int_{\mathbf{f}_*} p(\mathbf{x}', \mathbf{y}'|\mathbf{z})p(\mathbf{z}|X, Y, \mathbf{x}_*)d\mathbf{z}}, \quad (3.50)$$

where the previous posterior $p(\mathbf{f}_*|X, Y, \mathbf{x}_*)$ becomes the prior in the update. Hence the GP learns over time as model parameters (which are an input to the GP) are updated through pricing model recalibration.

4 Massively Scalable Gaussian Processes

Massively scalable Gaussian processes (MSGP) are a significant extension of the basic kernel interpolation framework described above. The core idea of the framework, which is detailed in (Gardner et al. 2018), is to improve scalability by combining GPs with “inducing point methods.” The basic setup is as follows; Using structured kernel interpolation (SKI), a small set of m inducing points are carefully selected from the original training points. The covariance matrix has a Kronecker and Toeplitz structure, which is exploited by the Fast Fourier Transform (FFT). Finally, output over the original input points is interpolated from the output at the

inducing points. The interpolation complexity scales linearly with dimensionality p of the input data by expressing the kernel interpolation as a product of 1D kernels. Overall, SKI gives $O(pn + pm\log m)$ training complexity and $O(1)$ prediction time per test point.

4.1 Structured Kernel Interpolation (SKI)

Given a set of m inducing points, the $n \times m$ cross-covariance matrix, $K_{X,U}$, between the training inputs, X , and the inducing points, \mathbf{U} , can be approximated as $\tilde{K}_{X,U} = W_X K_{U,U}$ using a (potentially sparse) $n \times m$ matrix of interpolation weights, W_X . This allows to approximate $K_{X,Z}$ for an arbitrary set of inputs Z as $K_{X,Z} \approx \tilde{K}_{X,U} W_Z^\top$. For any given kernel function, K , and a set of inducing points, \mathbf{U} , *structured kernel interpolation* (SKI) procedure (Gardner et al. 2018) gives rise to the following approximate kernel:

$$K_{\text{SKI}}(\mathbf{x}, \mathbf{z}) = W_X K_{U,U} W_z^\top, \quad (3.51)$$

which allows to approximate $K_{X,X} \approx W_X K_{U,U} W_X^\top$. Gardner et al. (2018) note that standard inducing point approaches, such as subset of regression (SoR) or fully independent training conditional (FITC), can be reinterpreted from the SKI perspective. Importantly, the efficiency of SKI-based MSGP methods comes from, first, a clever choice of a set of inducing points which exploit the algebraic structure of $K_{U,U}$, and second, from using very sparse local interpolation matrices. In practice, local cubic interpolation is used.

4.2 Kernel Approximations

If inducing points, U , form a regularly spaced P -dimensional grid, and we use a stationary product kernel (e.g., the RBF kernel), then $K_{U,U}$ decomposes as a Kronecker product of Toeplitz matrices:

$$K_{U,U} = \mathbf{T}_1 \otimes \mathbf{T}_2 \otimes \cdots \otimes \mathbf{T}_P. \quad (3.52)$$

The Kronecker structure allows one to compute the eigendecomposition of $K_{U,U}$ by separately decomposing $\mathbf{T}_1, \dots, \mathbf{T}_P$, each of which is much smaller than $K_{U,U}$. Further, a Toeplitz matrix can be approximated by a circulant matrix⁵ which eigendecomposes by simply applying a discrete Fourier transform (DFT) to its

⁵Gardner et al. (2018) explored 5 different approximation methods known in the numerical analysis literature.

first column. Therefore, an approximate eigendecomposition of each $\mathbf{T}_1, \dots, \mathbf{T}_P$ is computed via the FFT in only $O(m \log m)$ time.

4.2.1 Structure Exploiting Inference

To perform inference, we need to solve $(K_{\text{SKI}} + \sigma_n^2 I)^{-1} \mathbf{y}$; kernel learning requires evaluating $\log \det(K_{\text{SKI}} + \sigma_n^2 I)$. The first task can be accomplished by using an iterative scheme—linear conjugate gradients—which depends only on matrix vector multiplications with $(K_{\text{SKI}} + \sigma_n^2 I)$. The second is performed by exploiting the Kronecker and Toeplitz structure of $K_{U,U}$ for computing an approximate eigendecomposition, as described above.

In this chapter, we primarily use the basic interpolation approach for simplicity. However for completeness, Sect. 5.3 shows the scaling of the time taken to train and predict with MSGPs.

5 Example: Pricing and Greeking with Single-GPs

In the following example, the portfolio holds a long position in both a European call and a put option struck on the same underlying, with $K = 100$. We assume that the underlying follows Heston dynamics:

$$\frac{dS_t}{S_t} = \mu dt + \sqrt{V_t} dW_t^1, \quad (3.53)$$

$$dV_t = \kappa(\theta - V_t)dt + \sigma \sqrt{V_t} dW_t^2, \quad (3.54)$$

$$d\langle W^1, W^2 \rangle_t = \rho dt, \quad (3.55)$$

where the notation and fixed parameter values used for experiments are given in Table 3.1 under $\mu = r_0$. We use a Fourier Cosine method (Fang and Oosterlee 2008) to generate the European Heston option price training and testing data for the GP. We also use this method to compare the GP Greeks, obtained by differentiating the kernel function.

Table 3.1 lists the values of the parameters for the Heston dynamics and terms of the European Call and Put option contract used in our numerical experiments. Table 3.2 shows the values for the Euler time stepper used for simulating Heston dynamics and the credit risk model.

For each pricing time t_i , we simultaneously fit a multi-GP to both gridded call and put prices over stock price S and volatility \sqrt{V} , keeping time to maturity fixed. Figure 3.3 shows the gridded call (top) and put (bottom) price surfaces at various time to maturities, together with the GP estimate. Within each column in the figure, the same GP model has been simultaneously fitted to both the call and put price

Table 3.1 This table shows the values of the parameters for the Heston dynamics and terms of the European Call and Put option contracts

Parameter description	Symbol	Value
Mean reversion rate	κ	0.1
Mean reversion level	θ	0.15
Vol. of Vol.	σ	0.1
Risk-free rate	r_0	0.002
Strike	K	100
Maturity	T	2.0
Correlation	ρ	-0.9

Table 3.2 This table shows the values for the Euler time stepper used for market risk factor simulation

Parameter description	Symbol	Value
Number of simulation	M	1000
Number of time steps	n_s	100
Initial stock price	S_0	100
Initial variance	V_0	0.1

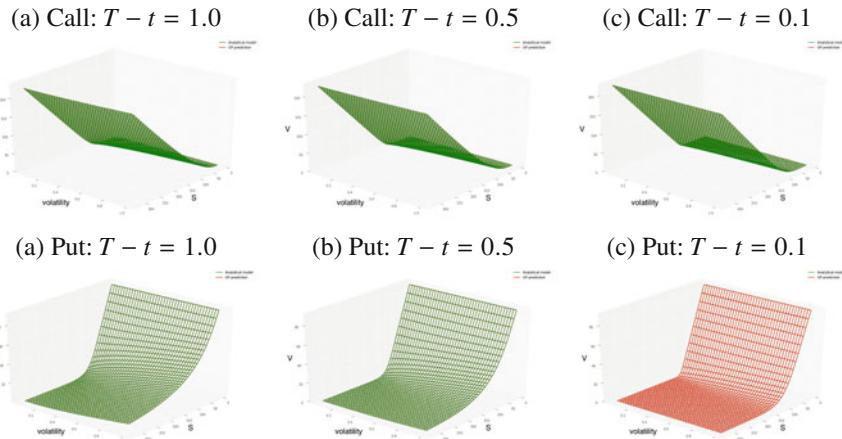


Fig. 3.3 This figure compares the gridded Heston model call (top) and put (bottom) price surfaces at various time to maturities, with the GP estimate. The GP estimate is observed to be practically identical (slightly below in the first five panels and slightly above in the last one). Within each column in the figure, the same GP model has been simultaneously fitted to both the Heston model call and put price surfaces over a 30×30 grid of prices and volatilities, fixing the time to maturity. Across each column, corresponding to different time to maturities, a different GP model has been fitted. The GP is then evaluated out-of-sample over a 40×40 grid, so that many of the test samples are new to the model. This is repeated over various time to maturities

surfaces over a 30×30 grid $\Omega_h \subset \Omega := [0, 1] \times [0, 1]$ of prices and volatilities,⁶ fixing the time to maturity. The scaling to the unit domain is not essential. However, we observed superior numerical stability when scaling.

⁶Note that the plot uses the original coordinates and not the re-scaled coordinates.

Across each column, corresponding to different time to maturities, a different GP model has been fitted. The GP is then evaluated out-of-sample over a 40×40 grid $\Omega_{h'} \subset \Omega$, so that many of the test samples are new to the model. This is repeated over various time to maturities.⁷

Extrapolation

One instance where kernel combination is useful in derivative modeling is for extrapolation—the appropriate mixture or combination of kernels can be chosen so that the GP is able to predict outside the domain of the training set. Noting that the payoff is linear when a call or put option is respectively deeply in and out-of-the money, we can configure a GP as a combination of a linear kernel and, say, a SE kernel. The linear kernel is included to ensure that prediction outside the domain preserves the linear property, whereas the SE kernel captures non-linearity. Figure 3.4 shows the results of using this combination of kernels to extrapolate the prices of a call struck at 110 and a put struck at 90. The linear property of the payoff function is preserved by the GP prediction and the uncertainty increases as the test point is further from the training set.

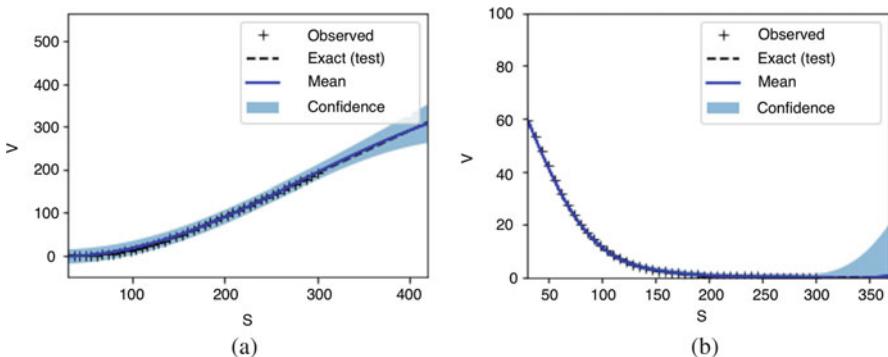


Fig. 3.4 This figure assesses the GP option price prediction in the setup of a Black–Scholes model. The GP with a mixture of a linear and SE kernel is trained on $n = 50 X, Y$ pairs, where $X \in \Omega^h \subset (0, 300]$ is the gridded underlying of the option prices and Y is a vector of call or put prices. These training points are shown by the black “+” symbols. The exact result using the Black–Scholes pricing formula is given by the black line. The predicted mean (blue solid line) and variance of the posterior are estimated from Eq. 3.44 over $m = 100$ gridded test points, $X_* \in \Omega_*^h \subset [300, 400]$, for the (left) call option struck at 110 and (center) put option struck at 90. The shaded envelope represents the 95% confidence interval about the mean of the posterior. This confidence interval is observed to increase the further the test point is from the training set. The time to maturity of the options are fixed to two years. (a) Call price. (b) Put price

⁷Such maturities might correspond to exposure evaluation times in CVA simulation as in Crépey and Dixon (2020). The option model versus GP model are observed to produce very similar values.

5.1 Greeking

The GP provides analytic derivatives with respect to the input variables

$$\partial_{X_*} \mathbb{E}[\mathbf{f}_* | X, Y, X_*] = \partial_{X_*} \boldsymbol{\mu}_{X_*} + \partial_{X_*} K_{X_*, X} \boldsymbol{\alpha}, \quad (3.56)$$

where $\partial_{X_*} K_{X_*, X} = \frac{1}{\ell^2} (X - X_*) K_{X_*, X}$ and we recall from Sect. (3.46) that $\boldsymbol{\alpha} = [K_{X,X} + \sigma_n^2 I]^{-1} \mathbf{y}$ (and in the numerical experiments we set $\boldsymbol{\mu} = 0$). Second-order sensitivities are obtained by differentiating once more with respect to X_* .

Note that $\boldsymbol{\alpha}$ is already calculated at training time (for pricing) by Cholesky matrix factorization of $[K_{X,X} + \sigma_n^2 I]$ with $O(n^3)$ complexity, so there is no significant computational overhead from Greeking. Once the GP has learned the derivative prices, Eq. 3.56 is used to evaluate the first order MtM Greeks with respect to the input variables over the test set. Example source code illustrating the implementation of this calculation is presented in the notebook `Example-2-GP-BS-Derivatives.ipynb`.

Figure 3.5 shows (left) the GP estimate of a call option's delta $\Delta := \frac{\partial C}{\partial S}$ and (right) vega $\nu := \frac{\partial C}{\partial \sigma}$, having trained on the underlying, respectively implied volatility, and on the BS option model prices. For avoidance of doubt, the model is not trained on the BS Greeks. For comparison in the figure, the BS delta and vega are also shown. In each case, the two graphs are practically indistinguishable, with one graph superimposed over the other.

5.2 Mesh-Free GPs

The above numerical examples have trained and tested GPs on uniform grids. This approach suffers from the curse of dimensionality, as the number of training points grows exponentially with the dimensionality of the data. This is why, in order to estimate the MtM cube, we advocate divide-and-conquer, i.e. the use of numerous

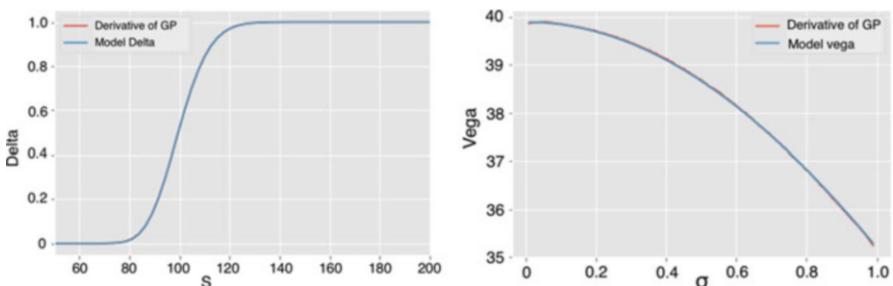


Fig. 3.5 This figure shows (left) the GP estimate of the call option's delta $\Delta := \frac{\partial C}{\partial S}$ and (right) vega $\nu := \frac{\partial C}{\partial \sigma}$, having trained on the underlying, respectively implied volatility, and on the BS option model prices

low input dimensional space, p , GPs run in parallel on specific asset classes. However, use of fixed grids is by no means necessary. We show here how GPs can show favorable approximation properties with a relatively few number of simulated reference points (cf. also (Gramacy and Apley 2015)).

Figure 3.6 shows the predicted Heston call prices using (left) 50 and (right) 100 simulated training points, indicated by “+”s, drawn from a uniform random distribution. The Heston call option is struck at $K = 100$ with a maturity of $T = 2$ years.

Figure 3.7 (left) shows the convergence of the GP MSE of the prediction, based on the number of Heston simulated training points. Fixing the number of simulated points to 100, but increasing the input space dimensionality, p , of each observation point (to include varying Heston parameters, Fig. 3.7 (right) shows the wall-clock time for training a GP with SKI (see Sect. 3.4). Note that the number of SGD iterations has been fixed to 1000.

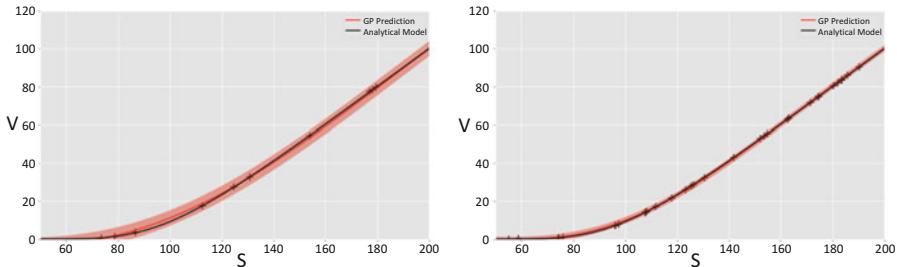


Fig. 3.6 Predicted Heston Call prices using (left) 50 and (right) 100 simulated training points, indicated by “+”s, drawn from a uniform random distribution

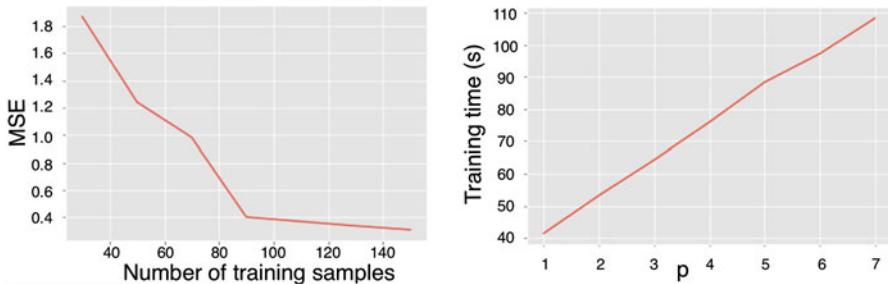


Fig. 3.7 (Left) The convergence of the GP MSE of the prediction is shown based on the number of simulated Heston training points. (Right) Fixing the number of simulated points to 100, but increasing the dimensionality p of each observation point (to include varying Heston parameters), the figure shows the wall-clock time for training a GP with SKI

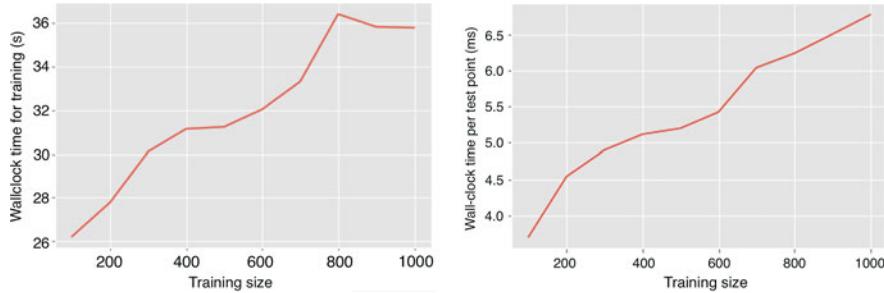


Fig. 3.8 (Left) The elapsed wall-clock time is shown for training against the number of training points generated by a Black–Scholes model. (Right) The elapsed wall-clock time for prediction of a single point is shown against the number of testing points. The reason that the prediction time increases (whereas the theory reviewed in Sect. 3.4 says it should be constant) is due to memory latency in our implementation—each point prediction involves loading a new test point into memory

5.3 Massively Scalable GPs

Figure 3.8 shows the increase of MSGP training time and prediction time against the number of training points n from a Black Scholes model. Fixing the number of inducing points to $m = 30$ (see Sect. 3.4), we increase the number of observations, n , in the $p = 1$ dimensional training set.

Setting the number of SGD iterations to 1000, we observe an approximate 1.4x increase in training time for a 10x increase in the training sample. We observe an approximate 2x increase in prediction time for a 10x increase in the training sample. The reason that the prediction time does not scale independently of n is due to memory latency in our implementation—each point prediction involves loading a new test point into memory. Fast caching approaches can be used to reduce this memory latency, but are beyond the scope of this section.

Note that training and testing times could be improved with CUDA on a GPU, but are not evaluated here.

6 Multi-response Gaussian Processes

A multi-output Gaussian process is a collection of random vectors, any finite number of which have a matrix-variate Gaussian distribution. We borrow from Chen et al. (2017) the following formulation of a separable multi-output kernel specification as per (Alvarez et al. 2012, Eq. (9)):

Definition (MGP) \mathbf{f} is a d variate Gaussian process on \mathbb{R}^p with vector-valued mean function $\mu : \mathbb{R}^p \mapsto \mathbb{R}^d$, kernel $k : \mathbb{R}^p \times \mathbb{R}^p \mapsto \mathbb{R}$, and positive semi-definite

parameter covariance matrix $\Omega \in \mathbb{R}^{d \times d}$, if the vectorization of any finite collection of vectors $\mathbf{f}(\mathbf{x}_1), \dots, \mathbf{f}(\mathbf{x}_n)$ have a joint multivariate Gaussian distribution,

$$\text{vec}([\mathbf{f}(\mathbf{x}_1), \dots, \mathbf{f}(\mathbf{x}_n)]) \sim \mathcal{N}(\text{vec}(M), \Sigma \otimes \Omega),$$

where $\mathbf{f}(\mathbf{x}_i) \in \mathbb{R}^d$ is a column vector whose components are the functions $\mathbf{f}_l(\mathbf{x}_i)\}_{l=1}^d$, M is a matrix in $\mathbb{R}^{d \times n}$ with $M_{li} = \mu_l(\mathbf{x}_i)$, Σ is a matrix in $\mathbb{R}^{n \times n}$ with $\Sigma_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, and $\Sigma \otimes \Omega$ is the Kronecker product

$$\begin{pmatrix} \Sigma_{11}\Omega & \cdots & \Sigma_{1n}\Omega \\ \vdots & \ddots & \vdots \\ \Sigma_{m1}\Omega & \cdots & \Sigma_{mn}\Omega \end{pmatrix}.$$

Sometimes Σ is called the column covariance matrix while Ω is the row (or task) covariance matrix. We denote $\mathbf{f} \sim \mathcal{MGP}(\mathbf{m}\mu, k, \Omega)$. As explained after Eq. (10) in (Alvarez et al. 2012), the matrices Σ and Ω encode dependencies among the inputs, respectively outputs.

6.1 Multi-Output Gaussian Process Regression and Prediction

Given n pairs of observations $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, $\mathbf{x}_i \in \mathbb{R}^p$, $\mathbf{y}_i \in \mathbb{R}^d$, we assume the model $\mathbf{y}_i = \mathbf{f}(\mathbf{x}_i)$, $i \in \{1, \dots, n\}$, where $\mathbf{f} \sim \mathcal{MGP}(\mu, k', \Omega)$ with $k' = k(\mathbf{x}_i, \mathbf{x}_j) + \delta_{ij}\sigma_n^2$, in which σ_n^2 is the variance of the additive Gaussian noise. That is, the vectorization of the collection of functions $[\mathbf{f}(\mathbf{x}_1), \dots, \mathbf{f}(\mathbf{x}_n)]$ follows a multivariate Gaussian distribution

$$\text{vec}([\mathbf{f}(\mathbf{x}_1), \dots, \mathbf{f}(\mathbf{x}_n)]) \sim \mathcal{N}(\mathbf{0}, K' \otimes \Omega),$$

where K' is the $n \times n$ covariance matrix of which the (i, j) -th element $[K']_{ij} = k'(\mathbf{x}_i, \mathbf{x}_j)$.

To predict a new variable $\mathbf{f}_* = [\mathbf{f}_{*1}, \dots, \mathbf{f}_{*m}]$ at the test locations $X_* = [\mathbf{x}_{n+1}, \dots, \mathbf{x}_{n+m}]$, the joint distribution of the training observations $Y = [\mathbf{y}_1, \dots, \mathbf{y}_n]$ and the predictive targets \mathbf{f}_* are given by

$$\begin{bmatrix} Y \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{MN}\left(\mathbf{0}, \begin{bmatrix} K'(X, X) & K'(X_*, X)^T \\ K'(X_*, X) & K'(X_*, X_*) \end{bmatrix}, \Omega\right), \quad (3.57)$$

where $K'(X, X)$ is an $n \times n$ matrix of which the (i, j) -th element $[K'(X, X)]_{ij} = k'(x_i, x_j)$, $K'(X_*, X)$ is an $m \times n$ matrix of which the (i, j) -th element

$[K'(X_*, X)]_{ij} = k'(x_{n+i}, x_j)$, and $K'(X_*, X_*)$ is an $m \times m$ matrix with the (i, j) -th element $[K'(X_*, X_*)]_{ij} = k'(x_{n+i}, x_{n+j})$. Thus, taking advantage of conditional distribution of multivariate Gaussian process, the predictive distribution is:

$$p(\text{vec}(\mathbf{f}_*)|X, Y, X_*) = \mathcal{N}(\text{vec}(\hat{M}), \hat{\Sigma} \otimes \hat{\Omega}), \quad (3.58)$$

where

$$\hat{M} = K'(X_*, X)^T K'(X, X)^{-1} Y, \quad (3.59)$$

$$\hat{\Sigma} = K'(X_*, X_*) - K'(X_*, X)^T K'(X, X)^{-1} K'(X_*, X), \quad (3.60)$$

$$\hat{\Omega} = \Omega. \quad (3.61)$$

The hyperparameters and elements of the covariance matrix Ω are found by minimizing the negative log marginal likelihood of observations:

$$\mathcal{L}(Y|X, \lambda, \Omega) = \frac{nd}{2} \ln(2\pi) + \frac{d}{2} \ln |K'| + \frac{n}{2} \ln |\Omega| + \frac{1}{2} \text{tr}((K')^{-1} Y \Omega^{-1} Y^T). \quad (3.62)$$

Further details of the multi-GP are given in (Bonilla et al. 2007; Alvarez et al. 2012; Chen et al. 2017). The computational remarks made in Sect. 3.4 also apply here, with the additional comment that the training and prediction time also scale linearly (proportionally) with the number of dimensions d . Note that the task covariance matrix Ω is estimated via a d -vector factor \mathbf{b} by $\Omega = \mathbf{b}\mathbf{b}^T + \sigma_\Omega^2 I$ (where the σ_Ω^2 component corresponds to a standard white noise term). An alternative computational approach, which exploits separability of the kernel, is the one described in Section 6.1 of (Alvarez et al. 2012), with complexity $O(d^3 + n^3)$.

7 Summary

In this chapter we have introduced Bayesian regression and shown how it extends many of the concepts in the previous chapter. We develop kernel based machine learning methods, known as Gaussian processes, and demonstrate their application to surrogate models of derivative prices. The key learning points of this chapter are:

- Introduced Bayesian linear regression;
- Derived the posterior distribution and the predictive distribution;
- Described the role of the prior as an equivalent form of regularization in maximum likelihood estimation; and
- Developed Gaussian Processes for kernel based probabilistic modeling, with programming examples in derivative modeling.

8 Exercises

Exercise 3.1: Posterior Distribution of Bayesian Linear Regression

Consider the Bayesian linear regression model

$$y_i = \boldsymbol{\theta}^T X + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2), \quad \boldsymbol{\theta} \sim \mathcal{N}(\mu, \Sigma).$$

Show that the posterior over data \mathcal{D} is given by the distribution

$$\boldsymbol{\theta} | \mathcal{D} \sim \mathcal{N}(\mu', \Sigma'),$$

with moments:

$$\begin{aligned}\mu' &= \Sigma' \boldsymbol{\alpha} = (\Sigma^{-1} + \frac{1}{\sigma_n^2} X X^T)^{-1} (\Sigma^{-1} \mu + \frac{1}{\sigma_n^2} \mathbf{y}^T X) \\ \Sigma' &= A^{-1} = (\Sigma^{-1} + \frac{1}{\sigma_n^2} X X^T)^{-1}.\end{aligned}$$

Exercise 3.2: Normal Conjugate Distributions

Suppose that the prior is $p(\theta) = \phi(\theta; \mu_0, \sigma_0^2)$ and the likelihood is given by

$$p(x_{1:n} | \theta) = \prod_{i=1}^n \phi(x_i; \theta, \sigma^2),$$

where σ^2 is assumed to be known. Show that the posterior is also normal, $p(\theta | x_{1:n}) = \phi(\theta; \mu_{\text{post}}, \sigma_{\text{post}}^2)$, where

$$\mu_{\text{post}} = \frac{\sigma_0^2}{\frac{\sigma^2}{n} + \sigma_0^2} \bar{x} + \frac{\sigma^2}{\frac{\sigma^2}{n} + \sigma_0^2} \mu_0,$$

$$\sigma_{\text{post}}^2 = \frac{1}{\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}},$$

where $\bar{x} := \frac{1}{n} \sum_{i=1}^n x_i$.

Exercise 3.3: Prediction with GPs

Show that the predictive distribution for a Gaussian Process, with model output over a test point, \mathbf{f}_* , and assumed Gaussian noise with variance σ_n^2 , is given by

$$\mathbf{f}_* | \mathcal{D}, \mathbf{x}_* \sim \mathcal{N}(\mathbb{E}[\mathbf{f}_* | \mathcal{D}, \mathbf{x}_*], \text{var}[\mathbf{f}_* | \mathcal{D}, \mathbf{x}_*]),$$

where the moments of the posterior over X_* are

$$\begin{aligned}\mathbb{E}[\mathbf{f}_* | \mathcal{D}, X_*] &= \boldsymbol{\mu}_{X_*} + K_{X_*, X}[K_{X, X} + \sigma_n^2 I]^{-1} Y, \\ \text{var}[\mathbf{f}_* | \mathcal{D}, X_*] &= K_{X_*, X} - K_{X_*, X}[K_{X, X} + \sigma_n^2 I]^{-1} K_{X, X_*}.\end{aligned}$$

8.1 Programming Related Questions*

Exercise 3.4: Derivative Modeling with GPs

Using the notebook `Example-1-GP-BS-Pricing.ipynb`, investigate the effectiveness of a Gaussian process with RBF kernels for learning the shape of a European derivative (call) pricing function $V_t = f_t(S_t)$ where S_t is the underlying stock's spot price. The risk free rate is $r = 0.001$, the strike of the call is $K_C = 130$, the volatility of the underlying is $\sigma = 0.1$ and the time to maturity $\tau = 1.0$.

Your answer should plot the variance of the predictive distribution against the stock price, $S_t = s$, over a dataset consisting of $n \in \{10, 50, 100, 200\}$ gridded values of the stock price $s \in \Omega^h := \{i \Delta s \mid i \in \{0, \dots, n-1\}, \Delta s = 200/(n-1)\} \subseteq [0, 200]$ and the corresponding gridded derivative prices $V(s)$. Each observation of the dataset, $(s_i, v_i = f_t(s_i))$ is a gridded (stock, call price) pair at time t .

Appendix

Answers to Multiple Choice Questions

Question 1

Answer: 1,4,5.

Parametric Bayesian regression always treats the regression weights as random variables.

In Bayesian regression the data function $f(x)$ is only observed if the data is assumed to be noise-free. Otherwise, the function is not directly observed.

The posterior distribution of the parameters will only be Gaussian if both the prior and the likelihood function are Gaussian. The distribution of the likelihood function depends on the assumed error distribution.

The posterior distribution of the regression weights will typically contract with increasing data. The precision matrix grows with decreasing variance and hence the variance of the posterior shrinks with increasing data. There are exceptions if, for example, there are outliers in the data.

The mean of the posterior distribution depends on both the mean and covariance of the prior if it is Gaussian. We can see this from Eq. 3.19.

Question 2

Answer: 1, 2, 4. Prediction under a Bayesian linear model requires first estimating the moments of the posterior distribution of the parameters. This is because the prediction is the expected likelihood of the new data under the posterior distribution.

The predictive distribution is Gaussian only if the posterior and likelihood distributions are Gaussian. The product of Gaussian density functions is also Gaussian.

The predictive distribution does not depend on the weights in the models - it is marginalized out under the expectation w.r.t. the posterior distribution. The variance of the predictive distribution typically contracts with increasing training data because the variance of the posterior and the likelihood typically decreases with increasing training data.

Question 3

Answer: 2, 3, 4.

Gaussian Process regression is a Bayesian modeling approach but they do not assume that the data is Gaussian distributed, neither do they make such an assumption about the error.

Gaussian Processes place a probabilistic prior directly on the space of functions and model the posterior of the predictor using a parameterized kernel representation of the covariance matrix. Gaussian Processes are fitted to data by maximizing the evidence for the kernel parameters. However, it is not necessarily the case that the choice of kernel is effectively a hyperparameter that can be optimized. While this could be achieved in an ad hoc way, there are other considerations which dictate the choice of kernel concerning smoothness and ability to extrapolate.

Python Notebooks

A number of notebooks are provided in the accompanying source code repository, beyond the two described in this chapter. These notebooks demonstrate the use of Multi-GPs and application to CVA modeling (see Crépey and Dixon (2020) for details of these models). Further details of the notebooks are included in the README .md file.

References

- Alvarez, M., Rosasco, L., & Lawrence, N. (2012). Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning*, 4(3), 195–266.
- Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Berlin, Heidelberg: Springer-Verlag.
- Bonilla, E. V., Chai, K. M. A., & Williams, C. K. I. (2007). Multi-task Gaussian process prediction. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, USA (pp. 153–160). Curran Associates Inc.
- Chen, Z., Wang, B., & Gorban, A. N. (2017, March). Multivariate Gaussian and student- t process regression for multi-output prediction. *ArXiv e-prints*.
- Cousin, A., Maatouk, H., & Rullière, D. (2016). Kriging of financial term structures. *European Journal of Operational Research*, 255, 631–648.

- Crépey, S., & M. Dixon (2020). Gaussian process regression for derivative portfolio modeling and application to CVA computations. *Computational Finance*.
- da Barrosa, M. R., Salles, A. V., & de Oliveira Ribeiro, C. (2016). Portfolio optimization through kriging methods. *Applied Economics*, 48(50), 4894–4905.
- Fang, F., & Oosterlee, C. W. (2008). A novel pricing method for European options based on Fourier-cosine series expansions. *SIAM J. SCI. COMPUT.*
- Gardner, J., Pleiss, G., Wu, R., Weinberger, K., & Wilson, A. (2018). Product kernel interpolation for scalable Gaussian processes. In *International Conference on Artificial Intelligence and Statistics* (pp. 1407–1416).
- Gramacy, R., & D. Apley (2015). Local Gaussian process approximation for large computer experiments. *Journal of Computational and Graphical Statistics*, 24(2), 561–578.
- Hans Bühler, H., Gonon, L., Teichmann, J., & Wood, B. (2018). Deep hedging. *Quantitative Finance*. Forthcoming (preprint version available as arXiv:1802.03042).
- Hernandez, A. (2017). Model calibration with neural networks. *Risk Magazine* (June 1–5). Preprint version available at SSRN.2812140, code available at <https://github.com/Andres-Hernandez/CalibrationNN>.
- Liu, M., & Staum, J. (2010). Stochastic kriging for efficient nested simulation of expected shortfall. *Journal of Risk*, 12(3), 3–27.
- Ludkovski, M. (2018). Kriging metamodels and experimental design for Bermudan option pricing. *Journal of Computational Finance*, 22(1), 37–77.
- MacKay, D. J. (1998). Introduction to Gaussian processes. In C. M. Bishop (Ed.), *Neural networks and machine learning*. Springer-Verlag.
- Melkumyan, A., & Ramos, F. (2011). Multi-kernel Gaussian processes. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Two*, IJCAI'11 (pp. 1408–1413). AAAI Press.
- Micchelli, C. A., Xu, Y., & Zhang, H. (2006, December). Universal kernels. *J. Mach. Learn. Res.*, 7, 2651–2667.
- Murphy, K. (2012). *Machine learning: a probabilistic perspective*. The MIT Press.
- Neal, R. M. (1996). *Bayesian learning for neural networks*, Volume 118 of *Lecture Notes in Statistics*. Springer.
- Pillonetto, G., Dinuzzo, F., & Nicolao, G. D. (2010, Feb). Bayesian online multitask learning of Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(2), 193–205.
- Rasmussen, C. E., & Ghahramani, Z. (2001). Occam's razor. In *In Advances in Neural Information Processing Systems 13* (pp. 294–300). MIT Press.
- Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. MIT Press.
- Roberts, S., Osborne, M., Ebden, M., Reece, S., Gibson, N., & Aigrain, S. (2013). Gaussian processes for time-series modelling. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 371(1984).
- Scholkopf, B., & Smola, A. J. (2001). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Cambridge, MA, USA: MIT Press.
- Spiegeleer, J. D., Madan, D. B., Reyners, S., & Schoutens, W. (2018). Machine learning for quantitative finance: fast derivative pricing, hedging and fitting. *Quantitative Finance*, 0(0), 1–9.
- Weinan, E., Han, J., & Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. arXiv:1706.04702.
- Whittle, P., & Sargent, T. J. (1983). *Prediction and regulation by linear least-square methods* (NED - New edition ed.). University of Minnesota Press.

Chapter 4

Feedforward Neural Networks



This chapter provides a more in-depth description of supervised learning, deep learning, and neural networks—presenting the foundational mathematical and statistical learning concepts and explaining how they relate to real-world examples in trading, risk management, and investment management. These applications present challenges for forecasting and model design and are presented as a reoccurring theme throughout the book. This chapter moves towards a more engineering style exposition of neural networks, applying concepts in the previous chapters to elucidate various model design choices.

1 Introduction

Artificial neural networks have a long history in financial and economic statistics. Building on the seminal work of (Gallant and White 1988; Andrews 1989; Hornik et al. 1989; Swanson and White 1995; Kuan and White 1994; Lo 1994; Hutchinson, Lo, and Poggio Hutchinson et al.; Baillie and Kapetanios 2007; Racine 2001) develop various studies in the finance, economics, and business literature. Most recently, the literature has been extended to include deep neural networks (Sirignano et al. 2016; Dixon et al. 2016; Feng et al. 2018; Heaton et al. 2017).

In this chapter we shall introduce some of the theory of function approximation and out-of-sample estimation with neural networks when the observation points are independent and typically also identically distributed. Such a case is not suitable for times series data and shall be the subject of later chapters. We shall restrict our attention to feedforward neural networks in order to explore some of the theoretical arguments which help us reason scientifically about architecture design and approximation error. Understanding these networks from a statistical, mathematical, and information-theoretic perspective is key to being able to successfully apply them in practice. While this chapter does present some simple financial examples to

highlight problematic conceptual issues, we defer the realistic financial applications to later chapters. Also, note that the emphasis of this chapter is how to build statistical models suitable for financial modeling, thus our emphasis is less on engineering considerations and more on how theory can guide the design of useful machine learning methods.

Chapter Objectives

By the end of this chapter, the reader should expect to accomplish the following:

- Develop mathematical reasoning skills to guide the design of neural networks;
 - Gain familiarity with the main theory supporting statistical inference with neural networks;
 - Relate feedforward neural networks with other types of machine learning methods;
 - Perform model selection with ridge and LASSO neural network regression;
 - Learn how to train and test a neural network; and
 - Gain familiarity with Bayesian neural networks.
-

Note that section headers ending with * are more mathematically advanced, often requiring some background in analysis and probability theory, and can be skipped by the less mathematically advanced reader.

2 Feedforward Architectures

2.1 Preliminaries

Feedforward neural networks are a form of supervised machine learning that use hierarchical layers of abstraction to represent high-dimensional non-linear predictors. The paradigm that deep learning provides for data analysis is very different from the traditional statistical modeling and testing framework. Traditional fit metrics, such as R^2 , t -values, p -values, and the notion of *statistical significance* has been replaced in the machine learning literature by out-of-sample forecasting and understanding the *bias-variance tradeoff*; that is the tradeoff between a more complex model versus over-fitting. Deep learning is data-driven and focuses on finding structure in large datasets. The main tools for variable or predictor selection are *regularization* and *dropout*.

There are a number of issues in any architecture design. How many layers? How many neurons N_l in each hidden layer? How to perform “variable selection?” Many of these problems can be solved by a stochastic search technique, called dropout

Srivastava et al. (2014), which we discuss in Sect. 5.2.2. Recall from Chap. 1 that a feedforward neural network model takes the general form of a parameterized map

$$Y = F_{W,b}(X) + \epsilon, \quad (4.1)$$

where $F_{W,b}$ is a deep neural network with L layers (Fig. 4.1) and ϵ is i.i.d. error. The deep neural network takes the form of a composition of simpler functions:

$$\hat{Y}(X) := F_{W,b}(X) = f_{W^{(L)}, b^{(L)}}^{(L)} \circ \cdots \circ f_{W^{(1)}, b^{(1)}}^{(1)}(X), \quad (4.2)$$

where $W = (W^{(1)}, \dots, W^{(L)})$ and $b = (b^{(1)}, \dots, b^{(L)})$ are weight matrices and bias vectors. Any weight matrix $W^{(\ell)} \in \mathbb{R}^{m \times n}$ can be expressed as n column m-vectors $W^{(\ell)} = [\mathbf{w}_{,1}^{(\ell)}, \dots, \mathbf{w}_{,n}^{(\ell)}]$. We denote each weight as $w_{ij}^{(\ell)} := [W^{(\ell)}]_{ij}$.

More formally and under additional restrictions, we can form this parameterized map in the class of compositions of *semi-affine functions*.

➤ Semi-Affine Functions

Let $\sigma : \mathbb{R} \rightarrow B \subset \mathbb{R}$ denote a continuous, monotonically increasing function whose codomain is a bounded subset of the real line. A function $f_{W^{(\ell)}, b^{(\ell)}}^{(\ell)} :$

(continued)

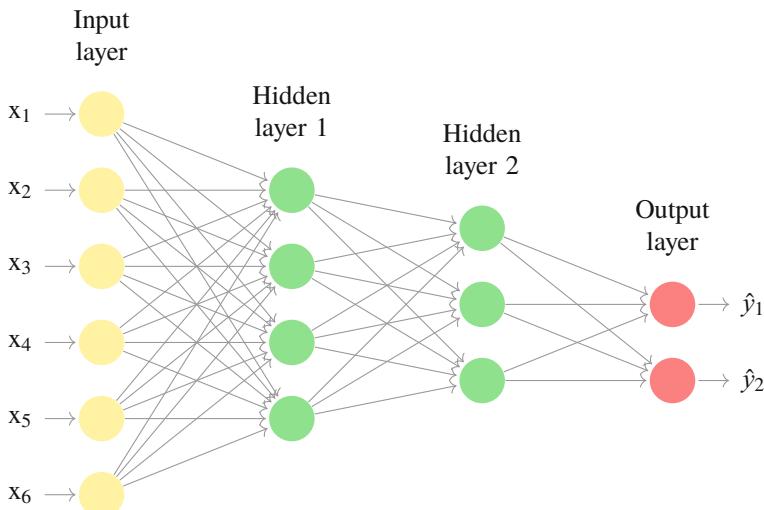


Fig. 4.1 An illustrative example of a feedforward neural network with two hidden layers, six features, and two outputs. Deep learning network classifiers typically have many more layers, use a large number of features and several outputs or classes. The goal of learning is to find the weight on every edge and the bias for every neuron (not illustrated) that minimizes the out-of-sample error

$\mathbb{R}^n \rightarrow \mathbb{R}^m$, given by $f(v) = W^{(\ell)}\sigma^{(\ell-1)}(v) + b^{(\ell)}$, $W^{(\ell)} \in \mathbb{R}^{m \times n}$ and $b^{(\ell)} \in \mathbb{R}^m$, is a semi-affine function in v , e.g. $f(v) = w \tanh(v) + b$. $\sigma(\cdot)$ are the activation functions of the output from the previous layer.

If all the activation functions are linear, $F_{W,b}$ is just linear regression, regardless of the number of layers L and the hidden layers are redundant. For any such network we can always find an equivalent network without hidden units. This follows from the fact that the composition of successive linear transformations is itself a linear transformation.¹ For example if there is one hidden layer and $\sigma^{(1)}$ is the identify function, then

$$\hat{Y}(X) = W^{(2)}(W^{(1)}X + b^{(1)}) + b^{(2)} = W^{(2)}W^{(1)}X + W^{(2)}b^{(1)} + b^{(2)} = \tilde{W}X + \tilde{b}. \quad (4.3)$$

Informally, the main effect of activation is to introduce non-linearity into the model, and in particular, interaction terms between the input. The geometric interpretation of the activation units will be discussed in the next section. We can view the special case when the network has one hidden layer and will see that the activation function introduces interaction terms $X_i X_j$. Consider the partial derivative

$$\partial_{X_j} \hat{Y} = \sum_i \mathbf{w}_{,i}^{(2)} \sigma'(I_i^{(1)}) w_{ij}^{(1)}, \quad (4.4)$$

where $\mathbf{w}_{,i}^{(2)}$ is the i th column vector of $W^{(2)}$, $I^{(\ell)}(X) := W^{(\ell)}X + b^{(\ell)}$, and differentiate again with respect to X_k , $k \neq i$ to give

$$\partial_{X_j, X_k}^2 \hat{Y} = -2 \sum_i \mathbf{w}_{,i}^{(2)} \sigma(I_i^{(1)}) \sigma'(I_i^{(1)}) w_{ij}^{(1)} w_{ik}^{(1)}, \quad (4.5)$$

which is not in general zero unless σ is the identity map.

2.2 Geometric Interpretation of Feedforward Networks

We begin by considering a simple feedforward binary classifier with only two features, as illustrated in Fig. 4.2. The simplest configuration we shall consider has just two inputs and one output unit—this is a multivariate regression model. More precisely, because we shall fit the model to binary responses, this network

¹Note that there is a potential degeneracy in this case; There may exist “flat directions”—hyper-surfaces in the parameter space that have exactly the same loss function.

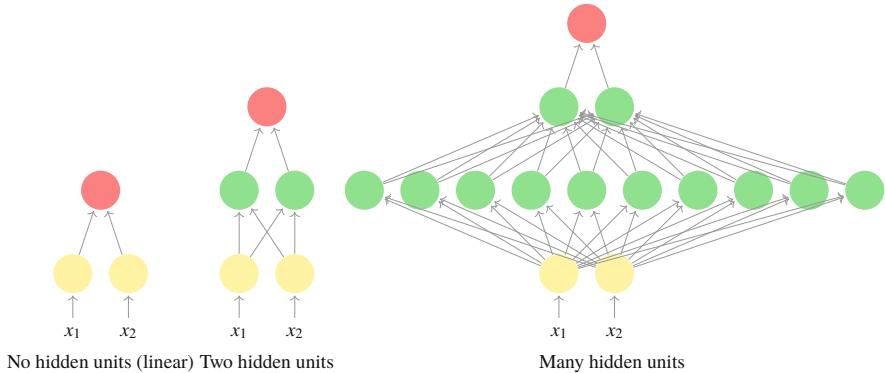


Fig. 4.2 Simple two variable feedforward networks with and without hidden layers. The yellow nodes denote input variables, the green nodes denote hidden units, and the red nodes are outputs. A feedforward network without hidden layers is a linear regressor. A feedforward network with one hidden layer is a *shallow learner* and a feedforward network with two or more hidden layers is a *deep learner*

is a logistic regression. Recall that only one output unit is required to represent the probability of a positive label, i.e. $\mathbb{P}[G = 1 | X]$. The next configuration we shall consider has one hidden layer—the number of hidden units shall be equal to the number of input neurons. This choice serves as a useful reference point as many hidden units are often needed for sufficient expressibility. The final configuration has substantially more hidden units. Note that the second layer has been introduced purely to visualize the output from the hidden layer. This set of simple configurations (a.k.a. architectures) is ample to illustrate how a neural network method works.

In Fig. 4.3 the data has been arranged so that no separating linear plane can perfectly separate the points in $[-1, 1] \times [-1, 1]$. The activation function is chosen to be $ReLU(x)$. The weight and biases of the network have been trained on this data. For each network, we can observe how the input space is transformed by the layers by viewing the top row of the figure. We can also view the linear regression in the original, input, space in the bottom row of the figure. The number of units in the first hidden layers is observed to significantly affect the classifier performance.²

Determining the weight and bias matrices, together with how many hidden units are needed for generalizable performance is the goal of parameter estimation and model selection. However, we emphasize that some conceptual understanding of neural networks is needed to derive interpretability, the topic of Chap. 5.

Partitioning

The partitioning of the input space is a distinguishing feature of neural networks compared to other machine learning methods. Each hidden unit defines a manifold

²There is some redundancy in the construction of the network and around 50 units are needed.

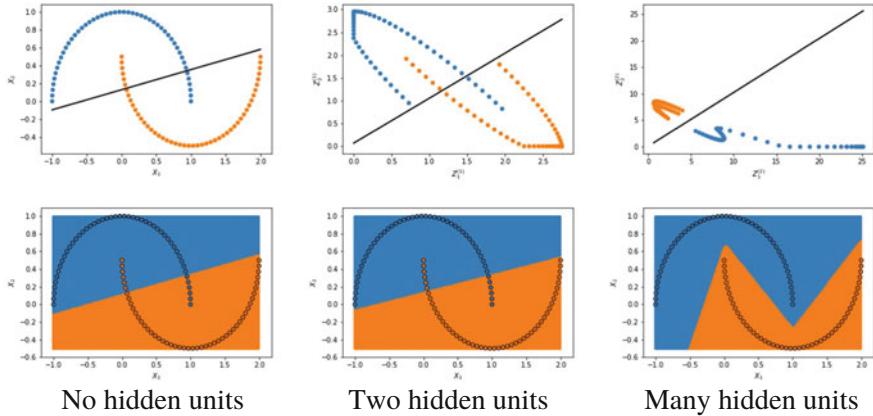


Fig. 4.3 This figure compares various feedforward neural network classifiers applied to a toy, non-linearly separable, binary classification dataset. Its purpose is to illustrate that increasing the number of hidden units in the first hidden layer provides substantial expressibility, even when the number of input variables is small. (Top) Each neural network classifier attempts to separate the labels with a hyperplane in the space of the output from the last hidden layer, $Z^{(L-1)}$. If the network has no hidden layers, then $Z^{(L-1)} = Z^{(0)} = X$. The features are shown in the space of $Z^{(L-1)}$. (Bottom) The separating hyperplane in the space of $Z^{(L-1)}$ is projected to the input space in order to visualize how the layers partition the input space. (Left) A feedforward classifier with no hidden layers is a logistic regression model—it partitions the input space with a plane. (Center) One hidden layer transforms the features by rotation, dilatation, and truncation. (Right) Two hidden layers with many hidden units perform an affine projection into high-dimensional space where points are more separable. See the Deep Classifiers notebook for an implementation of the classifiers and additional diagnostic tests (not shown here)

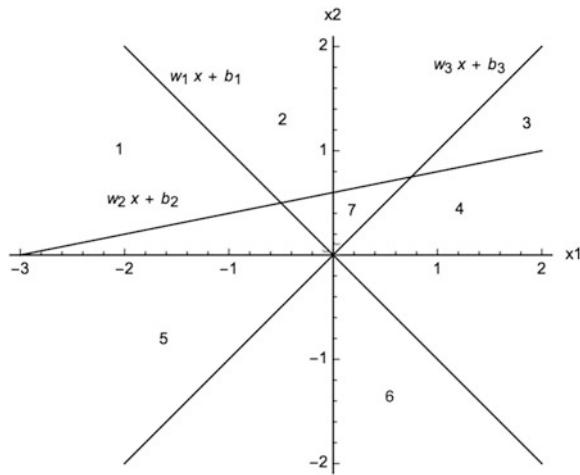
which divides the input space into convex regions. In other words, each unit in the hidden layer implements a half-space predictor. In the case of a ReLU activation function $f(x) = \max(x, 0)$, each manifold is simply a hyperplane and the neuron gets activated when the observation is on the “best” side of this hyperplane, the activation amount is equal to how far from the boundary the given point is. The set of hyperplanes defines a *hyperplane arrangement* (Montúfar et al. 2014). In general, an arrangement of $n \geq p$ hyperplanes in \mathbb{R}^p has at most $\sum_{j=0}^p \binom{n}{j}$ convex regions.

For example, in a two-dimensional input space, three neurons with ReLU activation functions will divide the space into no more than $\sum_{j=0}^2 \binom{3}{j} = 7$ regions, as shown in Fig. 4.4.

Multiple Hidden Layers

We can easily extend this geometrical interpretation to three-layered perceptrons ($L = 3$). Clearly, the neurons in the first (hidden) layer partition the network input space by corresponding hyperplanes into various half-spaces. Hence, the number of these half-spaces equals the number of neurons in the first layer. Then, the neurons in the second layer can classify the intersections of some of these half-spaces, i.e.

Fig. 4.4 Hyperplanes defined by three neurons in the hidden layer, each with ReLU activation functions, form a hyperplane arrangement. An arrangement of 3 hyperplanes in \mathbb{R}^2 has at most $\sum_{j=0}^2 \binom{3}{j} = 7$ convex regions



they represent convex regions in the input space. This means that a neuron from the second layer is active if and only if the network input corresponds to a point in the input space that is located simultaneously in all half-spaces, which are classified by selected neurons from the first layer.

The maximal number of linear regions of the functions computed by a neural network with p input units and $L - 1$ hidden layers, with equal width $n^{(\ell)} = n \geq p$ rectifiers at the ℓ th layer, can compute functions that have $\Omega\left(\left[\frac{n}{p}\right]^{(L-2)p} n^p\right)$ linear regions (Montúfar et al. 2014). We see that the number of linear regions of deep models grows exponentially in L and polynomially in n . See Montúfar et al. (2014) for a more detailed exposition of how the additional layers partition the input space.

While this form of reasoning guides our intuition towards designing neural network architectures it falls short at explaining why projection into a higher dimensional space is complementary to how the networks partition the input space. To address this, we turn to some informal probabilistic reasoning to aid our understanding.

2.3 Probabilistic Reasoning

Data Dimensionality

First consider any two independent standard Gaussian random p -vectors $X, Y \sim \mathcal{N}(0, I)$ and define their distance in Euclidean space by the 2-norm

$$d(X, Y)^2 := \|X - Y\|_2^2 = \sum_{i=1}^p (X_i - Y_i)^2. \quad (4.6)$$

Taking expectations gives

$$\mathbb{E}[d(X, Y)^2] = \sum_{i=1}^p \mathbb{E}[X_i^2] + \mathbb{E}[Y_i^2] = 2p. \quad (4.7)$$

Under these i.i.d. assumptions, the mean of the pairwise distance squared between any random points in \mathbb{R}^p is increasingly linear with the dimensionality of the space. By Jensen's inequality for concave functions, such as \sqrt{x}

$$\mathbb{E}[d(X, Y)] = \mathbb{E}[\sqrt{d(X, Y)^2}] \leq \sqrt{\mathbb{E}[d(X, Y)^2]} = \sqrt{2p}, \quad (4.8)$$

and hence the expected distance is bounded above by a function which grows to the power of $p^{1/2}$. This simple observation supports the characterization of random points as being less concentrated as the dimensionality of the input space increases. In particular, this property suggests machine learning techniques which rely on concentration of points in the input space, such as linear kernel methods, may not scale well with dimensionality. More importantly, this notion of loss of concentration with dimensionality of the input space does not conflict with how the input space is partitioned—the model defines a convex polytope with a less stringent requirement for locality of data for approximation accuracy.

Size of Hidden Layer

A similar simple probabilistic reasoning can be applied to the output from a one-layer network to understand how concentration varies with the number of units in the hidden layer. Consider, as before two i.i.d. random vectors X and Y in \mathbb{R}^p . Suppose now that these vectors are projected by a bounded semi-affine function $g : \mathbb{R}^p \rightarrow \mathbb{R}^q$. Assume that the output vectors $g(X), g(Y) \in \mathbb{R}^q$ are i.i.d. with zero mean and variance $\sigma^2 I$. Defining the distance between the output vectors as the 2-norm

$$d_g^2 := \|g(X) - g(Y)\|_2^2 = \sum_{i=1}^q (g_i(X) - g_i(Y))^2. \quad (4.9)$$

Under expectations

$$\mathbb{E}[d_g^2] = \sum_{i=1}^q \mathbb{E}[g(X)_i^2] + \mathbb{E}[g(Y)_i^2] = 2q\sigma^2 \leq q(\bar{g} - g) \quad (4.10)$$

and again by Jensen's inequality,

$$\mathbb{E}[d] \leq \sqrt{2}\sqrt{q}\sigma \leq \sqrt{q(\bar{g} - g)}, \quad (4.11)$$

we observe that the distance between the two output vectors, corresponding to the output of a hidden layer g under different inputs X and Y , can be less concentrated as the dimensionality of the output space increases. In other words, points in the codomain of g are on average more separate as q increases.

2.4 Function Approximation with Deep Learning*

While the above informal geometric and probabilistic reasoning provides some intuition for the need for multiple units in the hidden layer of a two-layer MLP, it does not address why deep networks are needed. The most fundamental mathematical concept in neural networks is the *universal representation* theorem. Simply put, this is a statement about the ability of a neural network to approximate any continuous, and unknown, function between input and output pairs with a simple, and known, functional representation. Hornik et al. (1989) show that a feedforward network with a single hidden layer can approximate any continuous function, regardless of the choice of activation function or data.

Formally, let $C^p := \{F : \mathbb{R}^p \rightarrow \mathbb{R} \mid F(x) \in C(\mathbb{R})\}$ be the set of continuous functions from \mathbb{R}^p to \mathbb{R} . Denote $\Sigma^p(g)$ as the class of functions $\{F : \mathbb{R}^p \rightarrow \mathbb{R} : F(x) = W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}\}$. Consider $\Omega = (0, 1]$ and let C_0 be the collection of all open intervals in $(0, 1]$. Then $\sigma(C_0)$, the σ -algebra generated by C_0 , is called the Borel σ -algebra. It is denoted by $\mathcal{B}((0, 1])$. An element of $\mathcal{B}((0, 1])$ is called a Borel set. A map $f : X \rightarrow Y$ between two topological spaces X, Y is called Borel measurable if $f^{-1}(A)$ is a Borel set for any open set A .

Let $M^p := \{F : \mathbb{R}^p \rightarrow \mathbb{R} \mid F(x) \in \mathcal{B}(\mathbb{R})\}$ be the set of all Borel measurable functions from \mathbb{R}^p to \mathbb{R} . We denote the Borel σ -algebra of \mathbb{R}^p as \mathcal{B}^p .

> Universal Representation Theorem

(Hornik et al. (1989)) For every monotonically increasing activation function σ , every input dimension size p , and every probability measure μ on $(\mathbb{R}^p, \mathcal{B}^p)$, $\Sigma^p(g)$ is uniformly dense on compacta in C^p and ρ_μ -dense in M^p .

This theorem shows that standard feedforward networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function arbitrarily well in the ρ_μ metric, regardless of the activation function (provided it is measurable), regardless of the dimension of the input space, p , and regardless of the input space. In other words, by taking the number of hidden units, k , *large enough*, every continuous function over \mathbb{R}^p can be approximated arbitrarily closely, uniformly over any bounded set by functions realized by neural networks with one hidden layer.

The universal approximation theorem is important because it characterizes feedforward networks with a single hidden layer as a class of approximate solutions. However, the theorem is not constructive—it does not specify how to configure a multilayer perceptron with the required approximation properties.

The theorem has some important limitations. It says nothing about the effect of adding more layers, other than to suggest they are redundant. It assumes that the optimal network weight vectors are reachable by gradient descent from the initial weight values, but this may not be possible in finite computations. Hence there are additional limitations introduced by the learning algorithm which are not apparent from a functional approximation perspective. The theorem cannot characterize the prediction error in any way, the result is purely based on approximation theory. An important concern is over-fitting and performance generalization on out-of-sample datasets, both of which it does not address. Moreover, it does not inform how MLPs can recover other approximation techniques, as a special case, such as polynomial spline interpolation. As such we shall turn to alternative theory in this section to assess the learnability of a neural network and to further understand it, beginning with a perceptron binary classifier. The reason why multiple hidden layers are needed is still an open problem, but various clues are provided in the next section and later in Sect. 2.7.

2.5 VC Dimension

In addition to expressive power, which determines the approximation error of the model, there is the notion of learnability, which determines the level of estimation error. The former measures the error introduced by an approximating function and the latter error measures the performance lost as a result of using a finite training sample.

One classical measure of the learnability of neural network classifiers is the Vapnik–Chervonenkis (VC) dimension. The VC dimension of a binary model $g = F_{W,b}(X)$ is the maximum number of points that can be arranged so that $F_{W,b}(X)$ shatters them, i.e. for all possible assignments of labels to those points, there exists a W, b such that $F_{W,b}$ makes no errors when classifying that set of data points. In the simplest case, a perceptron with n inputs units and a linear threshold activation $\sigma(x) := \text{sgn}(x)$ has a VC dimension of $n + 1$. For example, if $n = 1$, then

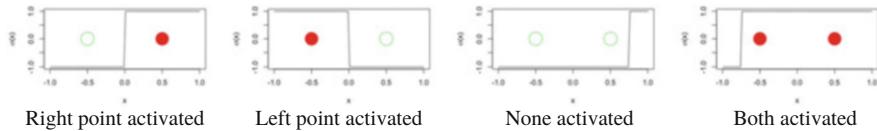


Fig. 4.5 For the points $\{-0.5, 0.5\}$, there are weights and biases that activate only one of them ($W = 1, b = 0$ or $W = -1, b = 0$), none of them ($W = 1, b = -0.75$), and both of them ($W = 1, b = 0.75$)

only two distinct points can always be correctly classified under all possible binary label assignments.

As shown in Fig. 4.5, for the points $\{-0.5, 0.5\}$, there are weights and biases that *activate* both of them ($W = 1, b = 0.75$), only one of them ($W = 1, b = 0$ or $W = -1, b = 0$), and none of them ($W = 1, b = -0.75$). Every distinct pair of points is separable with the linear threshold perceptron. So every dataset of size 2 is shattered by the perceptron. However, this linear threshold perceptron is incapable of shattering triplets, for example, $X \in \{-0.5, 0, 0.5\}$ and $Y \in \{0, 1, 0\}$. In general, the VC dimension of the class of half-spaces in \mathbb{R}^k is $k + 1$. For example, a 2d plane shatters any three points, but cannot shatter four points.

The VC dimension determines both the necessary and sufficient conditions for the consistency and rate of convergence of learning processes (i.e., the process of choosing an appropriate function from a given set of functions). If a class of functions has a finite VC dimension, then it is *learnable*. This measure of capacity is more robust than arbitrary measures such as the number of parameters. It is possible, for example, to find a simple set of functions that depends on only one parameter and that has infinite VC dimension.

? VC Dimension of an Indicator Function

Determine the VC dimension of the indicator function over $\Omega = [0, 1]$

$$\mathcal{F}(x) = \{f : \Omega \rightarrow \{0, 1\}, f(x) = \mathbb{1}_{x \in [t_1, t_2]}, \text{ or } f(x) = 1 - \mathbb{1}_{x \in [t_1, t_2]}, t_1 < t_2 \in \Omega\}. \quad (4.12)$$

Suppose there are three points x_1 , x_2 , and x_3 and assume $x_1 < x_2 < x_3$ without loss of generality. All possible labeling of the points is reachable; therefore, we assert that $VC(\mathcal{F}) \geq 3$. With four points x_1, x_2, x_3 , and x_4 (assumed increasing as always), you cannot label x_1 and x_3 with the value 1 and x_2 and x_4 with the value 0, for example. Hence $VC(\mathcal{F}) = 3$.

Recently (Bartlett et al. 2017a) prove upper and lower bounds on the VC dimension of deep feedforward neural network classifiers with the piecewise linear activation function, such as ReLU activation functions. These bounds are tight for almost the entire range of parameters. Letting $|W|$ be the number of weights and L be the number of layers, they proved that the VC dimension is $O(|W|L\log(|W|))$.

They further showed the effect of network depth on VC dimension with different non-linearities: there is *no dependence* for piecewise constant, *linear dependence* for piecewise-linear, and *no more than quadratic dependence* for general piecewise-polynomials.

Vapnik (1998) formulated a method of VC dimension based inductive inference. This approach, known as *structural empirical risk minimization*, achieved the smallest bound on the test error by using the training errors and choosing the machine (i.e., the set or functions) with the smallest VC dimension. The minimization problem expresses the bias-variance tradeoff. On the one hand, to minimize the bias, one needs to choose a function from a wide set of functions, not necessarily with a low VC dimension. On the other hand, the difference between the training error and the test error (i.e., variance) increases with VC dimension (a.k.a. expressibility).

The *expected risk* is an out-of-sample measure of performance of the learned model and is based on the joint probability density function (pdf) $p(x, y)$:

$$R[\hat{F}] = \mathbb{E}[\mathcal{L}(\hat{F}(X), Y)] = \int \mathcal{L}(\hat{F}(\mathbf{x}), y) dp(x, y). \quad (4.13)$$

If one could choose \hat{F} to minimize the expected risk, then one would have a definite measure of optimal learning. Unfortunately, the expected risk cannot be measured directly since this underlying pdf is unknown. Instead, we typically use the risk over the training set of N observations, also known as the *empirical risk measure* (ERM):

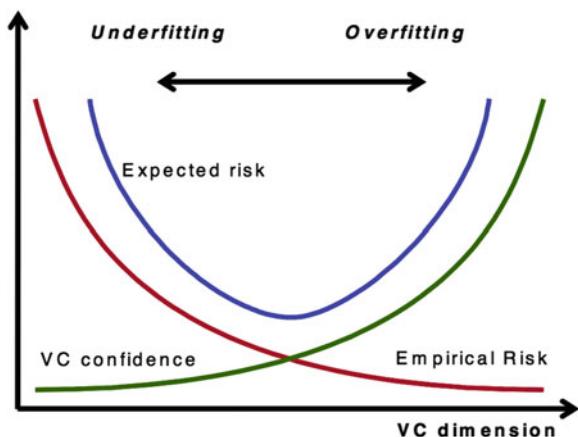
$$R_{emp}(\hat{F}) := \frac{1}{N} \sum_{i=1}^N \mathcal{L}(\hat{F}(\mathbf{x}_i), \mathbf{y}_i). \quad (4.14)$$

Under i.i.d. data assumptions, the law of large numbers ensures that the empirical risk will asymptotically converge to the expected risk. However, for small samples, one cannot guarantee that ERM will also minimize the expected risk. A famous result from statistical learning theory (Vapnik 1998) is that the VC dimension provides bounds on the expected risk as a function of the ERM and the number of training observations N , which holds with probability $(1 - \eta)$:

$$R[\hat{F}] \leq R_{emp}(\hat{F}) + \sqrt{\frac{h \left(\ln \left(\frac{2N}{h} \right) + 1 \right) - \ln \left(\frac{\eta}{4} \right)}{N}}, \quad (4.15)$$

where h is the VC dimension of $\hat{F}(X)$ and $N > h$. Figure 4.6 shows the tradeoff between VC dimension and the tightness of the bound. As the ratio N/h gets larger, i.e. for a fixed N , we decrease h , the VC confidence becomes smaller, and the actual risk becomes closer to the empirical risk. On the other hand, choosing a model with a higher VC dimension reduces the ERM at the expense of increasing the VC confidence.

Fig. 4.6 This figure shows the tradeoff between VC dimension and the tightness of the bound. As the ratio N/h gets larger, i.e. for a fixed N , we decrease h , the VC confidence becomes smaller, and the actual risk becomes closer to the empirical risk. On the other hand, choosing a model with a higher VC dimension reduces the ERM at the expense of increasing the VC confidence



The VC dimension plays a more dominant role in small-scale learning problems, where i.i.d. training data is limited and optimization error, that is the error introduced by the optimizer, is negligible. Beyond a certain sample size, computing power and the optimization algorithm become more dominant and the VC dimension is limited as a measure of learnability. Several studies demonstrate that VC dimension based error bounds are too weak and its usage, while providing some intuitive notion of model complexity, have faded in favor of alternative theories. Perhaps most importantly for finance, the bound in Eq. 4.15 only holds for i.i.d. data and little is known in the case when the data is auto-correlated.

? Multiple Choice Question 1

Which of the following statements are true:

1. The hidden units of a shallow feedforward network partition, with n hidden units, partition the input space in \mathbb{R}^p into no more than $\sum_{j=0}^p \binom{n}{j}$ convex regions.
2. The VC dimension of a Heaviside activated shallow feedforward network, with one hidden unit, and p features, is $p + 1$.
3. The bias-variance tradeoff is equivalently expressed through the VC confidence and the empirical risk measure.
4. The upper bound on the out-of-sample error of a feedforward network depends on its VC dimension and the number of training samples.
5. The VC dimension always grows linearly with the number of layers in a deep network.

2.6 When Is a Neural Network a Spline?*

Under certain choices of the activation function, we can construct MLPs which are a certain type of piecewise polynomial interpolants referred to as “splines.” Let $f(x)$ be any function whose domain is Ω and the function values $f_k := f(x_k)$ are known only at grid points $\Omega^h := \{x_k \mid x_k = kh, k \in \{1, \dots, K\}\} \subset \Omega \subset \mathbb{R}$ which are spaced by h . Note that the requirement that the data is gridded is for ease of exposition and is not necessary. We construct an orthogonal basis over Ω to give the interpolant

$$\hat{f}(x) = \sum_{i=1}^K \phi_i(x) f_i, \quad x \in \Omega, \quad (4.16)$$

where the $\{\phi_k\}_{k=1}^K$ are orthogonal basis functions. Under additional restrictions of the function space of f , we can derive error bounds which are a function of h .

We can easily demonstrate how a MLP with hidden units activated by Heaviside functions (unit step functions) is a piecewise constant functional approximation. Let $f(x)$ be any function whose domain is $\Omega = [0, 1]$. Suppose that the function is Lipschitz continuous, that is,

$$\forall x, x' \in [0, 1], \quad |f(x') - f(x)| \leq L|x' - x|,$$

for some constant $L \geq 0$. Using Heaviside functions to activate the hidden units

$$H(x) = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0, \end{cases} \quad (4.17)$$

we construct a neural network with $K = \lfloor \frac{L}{2\epsilon} + 1 \rfloor$ units in a single hidden layer that approximates $f(x)$ within $\epsilon > 0$. That is, $\forall x \in [0, 1]$, $|f(x) - \hat{f}(x)| \leq \epsilon$, where $\hat{f}(x)$ is the output of the neural network given input x . Let $\epsilon' = \frac{\epsilon}{L}$. We shall show that the neural network is a linear combination of indicator functions, ϕ_k , with compact support over $[x_k - \epsilon', x_k + \epsilon')$ and centered about x_k :

$$\phi_k(x) = \begin{cases} 1 & [x_k - \epsilon', x_k + \epsilon'), \\ 0 & \text{otherwise.} \end{cases} \quad (4.18)$$

The $\{\phi_k\}_{k=1}^K$ are piecewise constant basis functions, $\phi_i(x_j) = \delta_{ij}$, and the first few are illustrated in Fig. 4.7 below. The basis functions satisfy the partition of unity property $\sum_{k=1}^K \phi_k(x) = 1$, $\forall x \in \Omega$.

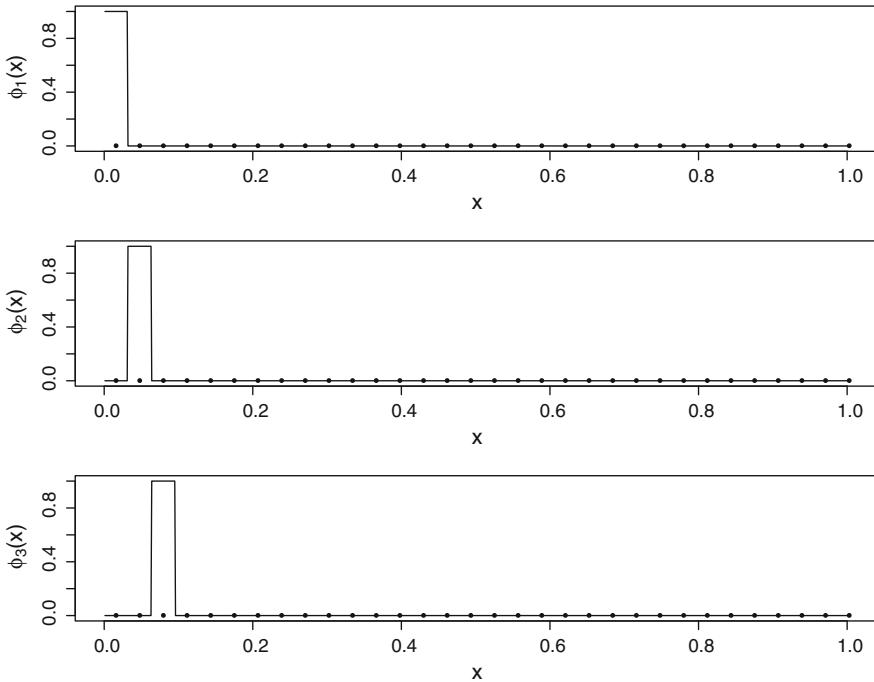


Fig. 4.7 The first three piecewise constant basis functions produced by the difference of neighboring step function activated units, $\phi_k(x) = H(x - (x_k - \epsilon')) - H(x - (x_k + \epsilon'))$

We shall construct such basis functions as the difference of Heaviside functions $\phi_k(x) = H(x - (x_k - \epsilon')) - H(x - (x_k + \epsilon'))$, $x_k = (2k - 1)\epsilon'$, by choosing the bias $b_k^{(1)} = -2(k - 1)\epsilon'$ and $W^{(1)} = \mathbf{1}$ so that the neural network, $\hat{f}(X) = W^{(2)}H(W^{(1)}X + b^{(1)})$ has values based on

$$H(W^{(1)}x + b^{(1)}) = \begin{bmatrix} H(x) \\ H(x - 2\epsilon') \\ \vdots \\ H(x - 2(k - 1)\epsilon') \\ \vdots \\ H(x - (2K - 1)\epsilon') \end{bmatrix}. \quad (4.19)$$

Then $W^{(2)}$ is set equal to exact function values and their differences:

$$W^{(2)} = [f(x_1), f(x_2) - f(x_1), \dots, f(x_K) - f(x_{K-1})], \quad (4.20)$$

so that

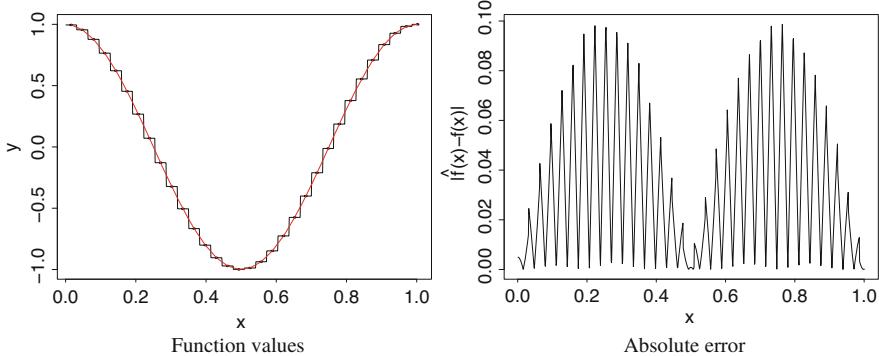


Fig. 4.8 The approximation of $\cos(2\pi x)$ using gridded input data and Heaviside activation functions. The error in approximation is at most ϵ with $K = \lfloor \frac{L}{2\epsilon} + 1 \rfloor$ hidden units

$$\hat{f} = \begin{cases} f(x_1), & x \leq 2\epsilon', \\ f(x_2), & 2\epsilon' < x \leq 4\epsilon', \\ \dots & \dots \\ f(x_k), & 2(k)\epsilon' < x \leq 2(k+1)\epsilon', \\ \dots & \dots \\ f(x_{K-1}), & 2(K-1)\epsilon' < x \leq 2K\epsilon'. \end{cases} \quad (4.21)$$

Figure 4.8 illustrates the function approximation for the case when $f(x) = \cos(2\pi x)$.

Since $x_k = (2k-1)\epsilon'$, we have that $\hat{Y} = f(x_k)$ over the interval $[x_k - \epsilon', x_k + \epsilon']$, which is the support of $\phi_k(x)$. By the Lipschitz continuity of $f(x)$, it follows that the worst-case error appears at the mid-point of any interval $[x_k, x_{k+1})$

$$|f(x_k + \epsilon') - \hat{f}(x_k + \epsilon')| = |f(x_k + \epsilon') - f(x_k)| \leq |f(x_k)| + L\epsilon' - |f(x_k)| = \epsilon. \quad (4.22)$$

This example is a special case of a more general representation permitted by MLPs. If we relax that the points need to be gridded, but instead just assume there are K data points in \mathbb{R}^p , then the region boundaries created by the K hidden units define a Voronoi diagram. Informally, a Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. The set of points are referred to as “seeds.” For each seed there is a corresponding region consisting of all points closer to that seed than to any other. The discussion of Voronoi diagrams is beyond the scope of this chapter, but suffice to say that the representation of MLPs as splines extends to higher dimensional input spaces and higher degree splines.

Hence, under a special configuration of the weights and biases, with the hidden units defining Voronoi cells for each observation, we can show that a neural network is a univariate spline. This result generalizes to higher dimensional and higher order splines. Such a result enables us to view splines as a special case of a neural network which is consistent with our reasoning of neural networks as generalized approximation and regression techniques. The formulation of neural networks as splines allows approximation theory to guide the design of the network. Unfortunately, equating neural networks with splines says little about why and when multiple layers are needed.

2.7 Why Deep Networks?

The extension to deep neural networks is in fact well motivated on statistical and information-theoretical grounds (Tishby and Zaslavsky 2015; Poggio 2016; Mhaskar et al. 2016; Martin and Mahoney 2018; Bartlett et al. 2017a). Poggio (2016) shows that deep networks can achieve superior performance versus linear additive models, such as linear regression, while avoiding the curse of dimensionality. There are additionally many recent theoretical developments which characterize the approximation behavior as a function of network depth, width, and sparsity level (Polson and Rockova 2018). Recently (Bartlett et al. 2017b) prove upper and lower bounds on the expressibility of deep feedforward neural network classifiers with the piecewise linear activation function, such as ReLU activation functions. These bounds are tight for almost the entire range of parameters. Letting n denote the total number of weights, they prove that the VC dimension is $O(nL\log(n))$.

> VC Dimension Theorem

Theorem (Bartlett et al. (2017b)) *There exists a universal constant C such that the following holds. Given any W, L with $W > CL > C^2$, there exists a ReLU network with $\leq L$ layers and $\leq W$ parameters with VC dimension $\geq WL\log(W/L)/C$. \square*

They further showed the effect of network depth on VC dimension with different non-linearities: there is no dependence for piecewise constant, linear dependence for piecewise-linear, and no more than quadratic dependence for general piecewise-polynomial. Thus the relationship between expressibility and depth is determined by the degree of the activation function. There is further ample theoretical evidence to

suggest that shallow networks cannot approximate the class of non-linear functions represented by deep ReLU networks without blow-up. Telgarsky (2016) shows that there is a ReLU network with L layers such that any network approximating it with only $O(L^{1/3})$ layers must have $\Omega(2^{L^{1/3}})$ units. Mhaskar et al. (2016) discuss the differences between composition versus additive models and show that it is possible to approximate higher polynomials much more efficiently with several hidden layers than a single hidden layer.

Martin and Mahoney (2018) shows that deep networks are implicitly self-regularizing behaving like Tikhonov regularization. Tishby and Zaslavsky (2015) characterizes the layers as “statistically decoupling” the input variables.

2.7.1 Approximation with Compositions of Functions

To gain some intuition as to why function composition can lead to successively more accurate function representation with each layer, consider the following example of a binary expansion of a decimal x .

Example 4.1 Binary Expansion of a Decimal

For each integer $n \geq 1$ and $x \in [0, 1]$, define $f_n(x) = x_n$, where x_n is the n th binary digit of x . The binary expansion of $x = \sum_{n=1}^{\infty} \frac{x_n}{2^n}$, where x_n is 1 or 0 depends on whether $X_{n-1} \geq \frac{1}{2^n}$ or otherwise, respectively, and $X_n := x - \sum_{i=1}^n \frac{x_i}{2^i}$. For example, we can find the first binary digit, x_1 as either 1 or 0 depending on whether $x_0 = x \geq \frac{1}{2}$. Now consider $X_1 = x - x_1/2$ and set $x_2 = 1$ if $X_1 \geq \frac{1}{2^2}$ or $x_2 = 0$ otherwise.

Example 4.2 Neural Network for Binary Expansion of a Decimal

A deep feedforward network for such a binary expansion of a decimal uses two neurons in each layer with different activations—Heaviside and identity functions. The input weight matrix, $W^{(1)}$, is the identity matrix, the other weight matrices, $\{W^{(\ell)}\}_{\ell>1}$ are

$$W^{(\ell)} = \begin{bmatrix} -\frac{1}{2^{\ell-1}} & 1 \\ -\frac{1}{2^{\ell-1}} & 1 \end{bmatrix},$$

and $\sigma_1^{(\ell)}(x) = H(x, \frac{1}{2^\ell})$ and $\sigma_2^{(\ell)}(x) = id(x) = x$. There are no bias terms. The output after ℓ hidden layers is the error, $X_\ell \leq \frac{1}{2^\ell}$.

While the example of converting a decimal in binary format using a binary expansion is simple, the approach can be readily generalized to the binary expansion of polynomials.

Theorem 4.2 (Liang and Srikant (2016)) *For the p th order polynomial $f(x) = \sum_{i=0}^p a_i x^i$, $x \in [0, 1]$ and $\sum_{i=1}^p |a_i| \leq 1$, there exists a multilayer neural network $\hat{f}(x)$ with $O(p + \log \frac{p}{\varepsilon})$ layers, $O(\log \frac{p}{\varepsilon})$ Heaviside units, and $O(p \log \frac{p}{\varepsilon})$ rectifier linear units such that $|f(x) - \hat{f}(x)| \leq \varepsilon$, $\forall x \in [0, 1]$.*

Proof The sketch of the proof is as follows. Liang and Srikant (2016) use the deep structure shown in Fig. 4.9 to find the n -step binary expansion $\sum_{i=0}^n a_i x^i$ of x . Then they construct a multilayer network to approximate polynomials $g_i(x) = x^i$, $i = 1, \dots, p$. Finally, they analyze the approximation error which is

$$|f(x) - \hat{f}(x)| \leq \frac{p}{2^{n-1}}.$$

See Appendix “Proof of Theorem 4.2” for the proof. \square

2.7.2 Composition with ReLU Activation

An intuitive way to understand the importance of multiple network layers is to consider the effect of composing piecewise affine functions instead of adding them. It is easy to see that combinations of ReLU activated neurons give piecewise affine

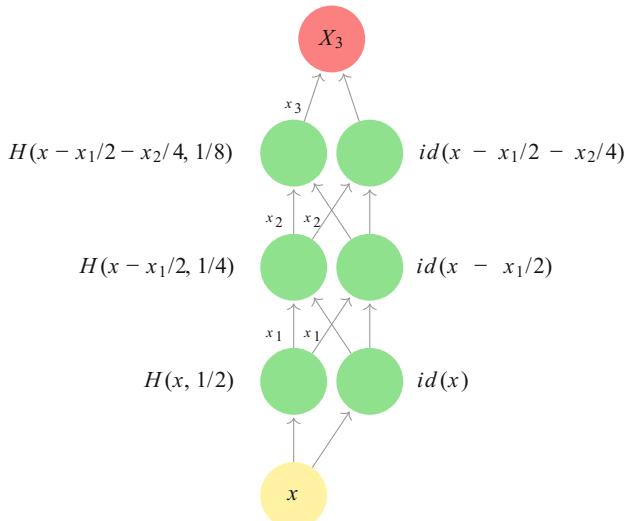


Fig. 4.9 An illustrative example of a deep feedforward neural network for binary expansion of a decimal. Each layer has two neurons with different activations—Heaviside and identity functions

approximations. For example, consider the shallow ReLU network with 2 and 4 perceptrons in Fig. 4.10:

$$F_{W,b} = W^{(2)}\sigma(W^{(1)}x + b^{(1)}), \quad \sigma := \max(x, 0).$$

Let us start by defining $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ to be *t-sawtooth* if it is piecewise affine with t pieces, meaning \mathbb{R} is partitioned into t consecutive intervals, and σ is affine within each interval. Consequently, $\text{ReLU}(x)$ is 2-sawtooth, but this class also includes many other functions, for instance, the decision stumps used in boosting are 2-sawtooth, and decision trees with $t - 1$ nodes correspond to t -sawtooths. The following lemma serves to build intuition about the effect of adding versus composing sawtooth functions which is illustrated in Fig. 4.11.

Lemma 4.1 *Let $f : \mathbb{R} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$ be, respectively, k - and l -sawtooth. Then $f + g$ is $(k + l)$ -sawtooth, and $f \circ g$ is kl -sawtooth.* \square

Fig. 4.10 A Shallow ReLU network with (left) two perceptrons and (right) four perceptrons

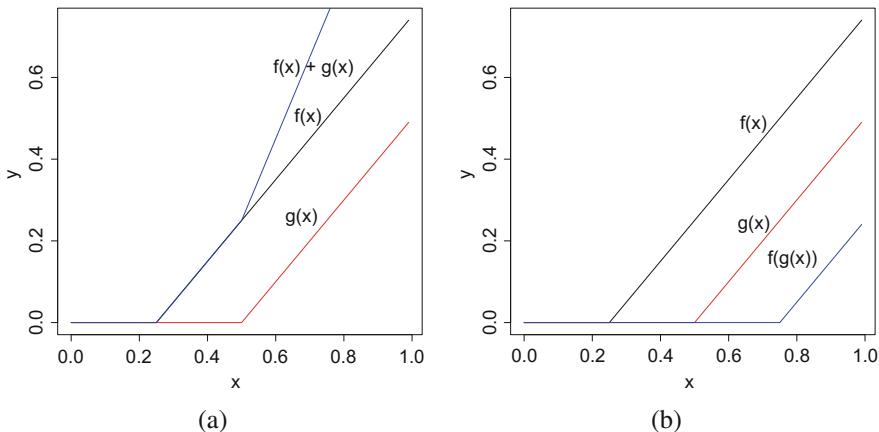
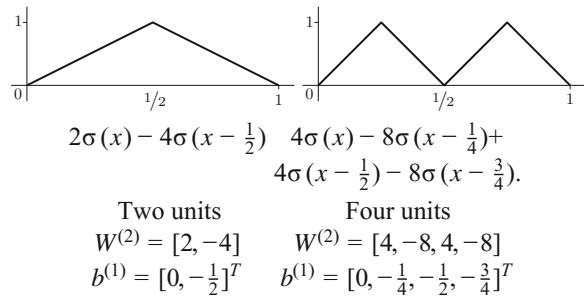


Fig. 4.11 Adding versus composing 2-sawtooth functions. (a) Adding 2-sawtooths. (b) Composing 2-sawtooths

Let us now build on this result by considering the *mirror map* $f_m : \mathbb{R} \rightarrow \mathbb{R}$, which is shown in Fig. 4.12, and defined as

$$f_m(x) := \begin{cases} 2x & \text{when } 0 \leq x \leq 1/2, \\ 2(1-x) & \text{when } 1/2 < x \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Note that f_m can be represented by a two-layer ReLU activated network with two neurons; For instance, $f_m(x) = (2\sigma(x) - 4\sigma(x-1/2))$. Hence f_m^k is the composition of k (identical) ReLU sub-networks. A key observation is that fewer hidden units are needed to shatter a set of points when the network is deep versus shallow.

Consider, for example, the sequence of $n = 2^k$ points with alternating labels, referred to as the n -ap, and illustrated in Fig. 4.13 for the case when $k = 3$. As the x values pass from left to right, the labels change as often as possible and provide the most challenging arrangement for shattering n points.

There are many ways to measure the representation power of a network, but we will consider the classification error here. Suppose that we have a σ activated network with m units per layer and l layers. Given a function $f : \mathbb{R}^p \rightarrow \mathbb{R}$ let $\tilde{f} : \mathbb{R}^p \rightarrow \{0, 1\}$ denote the corresponding classifier $\tilde{f}(x) := \mathbb{1}_{f(x) \geq 1/2}$, and additionally given a sequence of points $((x_i, y_i))_{i=1}^n$ with $x_i \in \mathbb{R}^p$ and $y_i \in \{0, 1\}$, define the classification error as $\mathcal{E}(f) := \frac{1}{n} \sum_i \mathbb{1}_{\tilde{f}(x_i) \neq y_i}$.

Given a sawtooth function, its classification error on the n -ap may be lower bounded as follows.

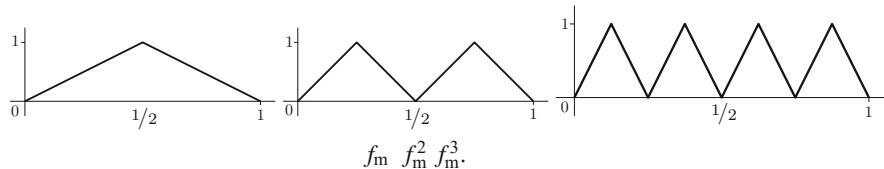


Fig. 4.12 The mirror map composed with itself

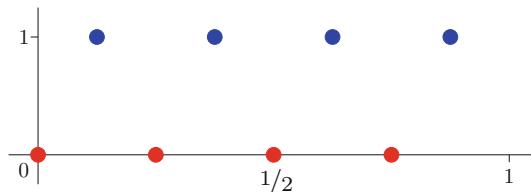


Fig. 4.13 The n -ap consists of n uniformly spaced points with alternating labels over the interval $[0, 1 - 2^{-n}]$. That is, the points $((x_i, y_i))_{i=1}^n$ with $x_i = i2^{-n}$ and $y_i = 0$ when i is even, and otherwise $y_i = 1$

Lemma 4.2 Let $((x_i, y_i))_{i=1}^n$ be given according to the n -ap. Then every t -sawtooth function $f : \mathbb{R} \rightarrow \mathbb{R}$ satisfies $\mathcal{E}(f) \geq (n - 4t)/(3n)$. \square

The proof in the appendix relies on a simple counting argument for the number of crossings of $1/2$. If there are m t -saw-tooth functions, then by Lemma 4.1, the resultant is a piecewise affine function over mt intervals. The main theorem now directly follows from Lemma 4.2.

Theorem 4.3 Let positive integer k , number of layers l , and number of nodes per layer m be given. Given a t -sawtooth $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ and $n := 2^k$ points as specified by the n -ap, then

$$\min_{W,b} \mathcal{E}(f) \geq \frac{n - 4(tm)^l}{3n}.$$

From this result one can say, for example, that on the n -ap one needs $m = 2^{k-3}$ many units when classifying with a ReLU activated shallow network versus only $m = 2^{(1/l(k-2)-1)}$ units per layer for a $l \geq 2$ deep network.

Research on deep learning is very active and there are still many questions that need to be addressed before deep learning is fully understood. However, the purpose of these examples is to build intuition and motivate the need for many hidden layers in addition to the effect of increasing the number of neurons in each hidden layer.

In the remaining part of this chapter we turn towards the practical application of neural networks and consider some of the primary challenges in the context of financial modeling. We shall begin by considering how to preserve the shape of functions being approximated and, indeed, how to train and evaluate a network.

3 Convexity and Inequality Constraints

It may be necessary to restrict the range of $\hat{f}(x)$ or impose certain properties which are known about the shape of the function $f(x)$ being approximated. For example, $V = f(S)$ might be an option price and S the value of the underlying asset and convexity and non-negativity of $\hat{f}(S)$ are necessary. Consider the following feedforward network architecture $F_{W,b}(X) : \mathbb{R}^p \rightarrow \mathbb{R}^d$:

$$\hat{Y} = F_{W,b}(X) = f_{W^{(L)}, b^{(L)}}^{(L)} \circ \cdots \circ f_{W^{(1)}, b^{(1)}}^{(1)}(X), \quad (4.23)$$

where

$$f_{W^{(\ell)}, b^{(\ell)}}^{(\ell)}(x) = \sigma(W^{(\ell)}x + b^{(\ell)}), \quad \forall \ell \in \{1, \dots, L\}. \quad (4.24)$$

Convexity

For convexity of \hat{Y} w.r.t. x , the activation function, $\sigma(x)$, must be a convex function of x . For avoidance of doubt, this convexity constraint should not be confused with convexity of the loss function w.r.t. the weights as in, for example, Bengio et al. (2006).

Examples³ include $\text{ReLU}(x) := \max(x, 0)$ and $\text{softplus}(x; t) := \frac{1}{t} \ln(1 + \exp\{tx\})$. For this class of activation functions, the semi-affine function $f_{W^{(\ell)}, b^{(\ell)}}^{(\ell)}(x) = \sigma(W^{(\ell)}x + b^{(\ell)})$ must also be convex in x since a convex function of a linear combination of x is also convex in x . The composition, $f_{W^{(\ell+1)}, b^{(\ell+1)}}^{(\ell+1)} \circ f_{W^{(\ell)}, b^{(\ell)}}^{(\ell)}(x)$, is convex if and only if $f_{W^{(\ell+1)}, b^{(\ell+1)}}^{(\ell+1)}(x)$ is non-decreasing convex and $f_{W^{(\ell)}, b^{(\ell)}}^{(\ell)}(x)$ is convex. The proof is left to the reader as a straightforward exercise. Hence, for convexity of $\hat{f}(x) = F_{W, b}(x)$ w.r.t. x we require that the weights in all but the first layer be positive:

$$w_{ij}^{(\ell)} \geq 0, \forall i, j, \forall \ell \in \{2, \dots, L\}. \quad (4.25)$$

The constraints on the weights needed to enforce convexity guarantee non-negative output if the bias $b_i^{(L)} \geq 0, \forall i \in \{1, \dots, d\}$ and $\sigma(x) \geq 0, \forall x$. Since $w_{ij}^{(L)} \geq 0, \forall i, j$ it follows that

$$w_{ij}^{(L)} \sigma(I_i^{(L-1)}) \geq 0, \quad (4.26)$$

and with non-negative bias terms, \hat{f}_i is non-negative.

We now separately consider bounding the network output independently of imposing convexity on $\hat{f}_i(x)$ w.r.t. x . If we choose a bounded activation function $\sigma \in [\underline{\sigma}, \bar{\sigma}]$, then we can easily impose linear inequality constraints to ensure that $\hat{f}_i \in [c_i, d_i]$

$$c_i \leq \hat{f}_i(x) \leq d_i, d_i > c_i, i \in \{1, \dots, d\}, \quad (4.27)$$

by setting

$$b_i^{(L)} = c_i - \sum_{j=1}^{n^{(L-1)}} \min(s_{ij}|\underline{\sigma}|, s_{ij}|\bar{\sigma}|)|w_{ij}^{(L)}|, s_{ij} := \text{sign}(w_{ij}^{(L)}). \quad (4.28)$$

³The parameterized softplus function $\sigma(x; t) = \frac{1}{t} \ln(1 + \exp\{tx\})$, with a model parameter $t \gg 1$, converges to the ReLU function in the limit $t \rightarrow \infty$.

Note that the expression inside the min function can be simplified further to $\min(s_{ij}|\underline{\sigma}|, s_{ij}|\bar{\sigma}|)|w_{ij}| = \min(w_{ij}|\underline{\sigma}|, w_{ij}|\bar{\sigma}|)$. Training of the weights and biases is a constrained optimization problem with the linear constraints

$$\sum_{j=1}^{n^{(L-1)}} \max(s_{ij}|\underline{\sigma}|, s_{ij}|\bar{\sigma}|)|w_{ij}^{(L)}| \leq d_i - b_i^{(L)}, \quad (4.29)$$

which can be solved with the method of Lagrange multipliers or otherwise. If we require that \hat{f}_i should be convex and bounded in the interval $[c_i, d_i]$, then the additional constraint, $w_{ij}^{(L)} \geq 0, \forall i, j$, is needed of course and the above simplifies to

$$b_i^{(L)} = c_i - \underline{\sigma} \sum_{j=1}^{n^{(L-1)}} w_{ij}^{(L)} \quad (4.30)$$

and solving the underdetermined system for $w_{ij}^{(L)}, \forall j$:

$$\sum_{j=1}^{n^{(L-1)}} \bar{\sigma} w_{ij}^{(L)} \leq d_i - b_i^{(L)} \quad (4.31)$$

$$\sum_{j=1}^{n^{(L-1)}} w_{ij}^{(L)} \leq \frac{d_i - c_i}{(\bar{\sigma} - \underline{\sigma})}. \quad (4.32)$$

The following toy examples provide simplified versions of constrained learning problems that arise in derivative modeling and calibration. The examples are intended only to illustrate the methodology introduced here. The first example is motivated by the need to learn an arbitrage free option price as a function of the underlying asset price. In particular, there are three scenarios where neural networks, and more broadly, supervised machine learning is useful for pricing. First, it provides a “model-free” framework, where no data generation process is assumed for the underlying dynamics. Second, machine learning can price complex derivatives where no analytic solution is known. Finally, machine learning does not suffer from the curse of dimensionality w.r.t. to the input space and can thus scale to basket options, options on many underlying assets. Each of these aspects merits further exploration and our example illustrates some of the challenges with learning pricing functions.

Perhaps the single largest defect of conventional derivative pricing models, however, is their calibration to data. Machine learning, too, provides an answer here—it provides a method for learning the relationship between market and contract variables and the model parameters.

Example 4.3 Approximating Option Prices

The payoff of a European call option at expiry time T is $V_T = \max(S_T - K, 0)$ and is convex with respect to S . Under the risk-neutral measure the option price at time t is the conditional expectation $V_t = \mathbb{E}_t[\exp\{-r(T-t)\}V_T]$. Since the conditional expectation is a linear operator, it preserves the convexity of the payoff function, so that the option price is always convex w.r.t. the underlying price. Thus, the second derivative, γ is always non-negative. Furthermore, the option price must always be non-negative.

Let us approximate the surface of a European call option with strike K over all underlying values $S_i \in (0, \bar{S})$. The input variable $X \in \mathbb{R}^+$ are underlying asset prices and the outputs are call prices, so that the data is $\{S_i, V_i\}$. We use a neural network to learn the relation $V = f(S)$ and enforce the property that f is non-negative and convex w.r.t. S .

In the following example, we train the MLP over a uniform grid of 100 training points $S_i \in \Omega_h \subset [0.001, 300]$, and $V_i = f(S_i)$ generated by the Black–Scholes (BS) pricing formula. The risk-free rate $r = 0.01$, the strike is 130, the volatility is σ , and time to maturity is $T = 2.0$. The test data of 100 observations are on a different uniform gridded over a wider domain $[0.001, 600]$. The network uses one hidden layer ($L = 2$) with 100 units, a softplus activation function, and $w_{ij}^{(L)}, b_i^{(L)} \geq 0, \forall i, j$. Figure 4.14 compares the prediction with the BS model over the test set. \hat{Y} is observed to be convex w.r.t. S because $w_{ij}^{(2)}$ is non-negative. Additionally, because $b_i^{(2)} \geq 0$ and $\underline{\sigma} = 0$, $\hat{Y} \geq 0$.

The figure also compares the Black–Scholes formula for the delta of the call option, $\Delta(X)$, the derivative of the price w.r.t. to S with the gradient of \hat{Y} :

$$\hat{\Delta}(X) = \partial_X \hat{Y} = (W^{(2)})^T D W^{(1)}, \quad D_{ii} = \frac{1}{1 + \exp\{-\mathbf{w}_i^{(1)} X - b_i^{(1)}\}}. \quad (4.33)$$

Under the BS model, the delta of a call option is in the interval $[0, 1]$. Note that the delta, although observed positive here, could be negative since there are no restrictions on $W^{(1)}$. Similarly, the delta approximation is observed to exceed unity. Thus, additional constraints are needed to bound the delta. For this architecture, imposing $w_{ij}^{(1)} \geq 0$ preserves the non-negativity of the delta and $\sum_j^{n^{(1)}} w_{ij}^{(2)} \mathbf{w}_j^{(1)} \leq 1, \forall i$ bounds the delta at unity.

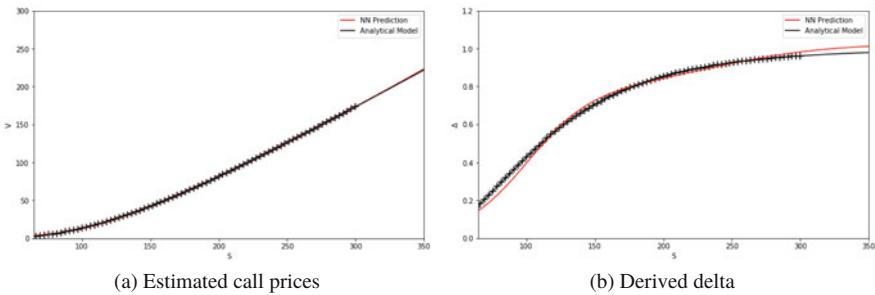


Fig. 4.14 (a) The out-of-sample call prices are estimated using a single-layer neural network with constraints to ensure non-negativity and convexity of the price approximation w.r.t. the underlying price S . (b) The analytic derivative of \hat{Y} is taken as the approximation of delta and compared over the test set with the Black–Scholes delta. We observe that additional constraints on the weights are needed to ensure that $\partial_X \hat{Y} \in [0, 1]$

Example 4.4 Calibrating Options

The goal is to learn the inverse of the Black–Scholes formula, as a function of moneyness, $M = S/K$. For simplicity, it considers the calibration of a chain of European in-the-money put or in-the-money equity call options with fixed time to maturity only. The input is moneyness for each option in the chain. The output of the neural network is the BS implied volatility—this is the implied volatility needed to calibrate the BS model to option price data corresponding to each moneyness.

The neural network preserves the positivity of the volatility and, in this example, imposes a convexity constraint on the surface w.r.t. to moneyness. The latter ensures consistency with liquid option markets, the implied volatility for both puts and calls typically monotonically increases as the strike price moves away from the current stock price—the so-called implied volatility smile. In markets, such as the equity markets, an implied volatility skew occurs because money managers usually prefer to write calls over puts.

The input variable $X \in \mathbb{R}^+$ is moneyness and the output is volatility so that the training data is $\{M_i, \sigma_i\}$. We use a neural network to learn the relation $\sigma = f(M)$ and enforce the property that f is non-negative and convex w.r.t. M . Note, in this example, that we do not directly learn the relationship between option prices and implied volatilities. Instead we learn how a BS root finder approximates the implied volatility as a function of the moneyness.

(continued)

Example 4.4 (continued)

In the following example, we train the MLP over a uniform grid of $n = 100$ training points $M_i \in \Omega_h \subset [0.5, 1 \times 10^4]$, and $\sigma_i = f(M_i)$ is generated by using a root finder for $V(\sigma; S, K_i, \tau, r) - \hat{V}_i = 0$, $\forall i = 1, \dots, n$ and $\tau = 0.2$ years using the option price with strike K_i and time to maturity τ . The risk-free rate $r = 0.01$. The test data of 100 observations are on a different uniform gridded over a wider domain $[0.4166, 1 \times 10^4]$. The network uses one hidden layer ($L = 2$) with 100 units, a softplus activation function, and $w_{ij}^{(L)}, b_i^{(L)} \leq 0$, $\forall i, j$. Figure 4.15 compares the out-of-sample model output with the root finder for the BS model over the test set. \hat{Y} is observed to be convex w.r.t. M because $w_{ij}^{(2)}$ is non-negative. Additionally, because $b_i^{(2)} \geq 0$ and $\underline{\sigma} = 0$, $\hat{Y} \geq 0$.

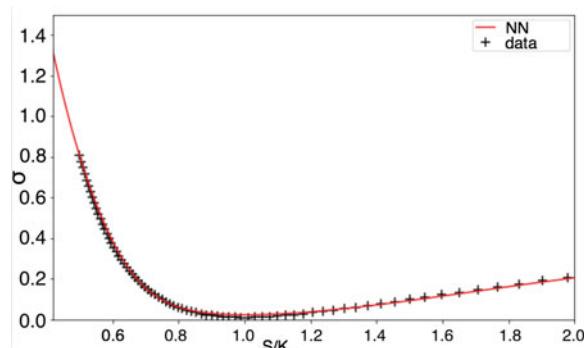
No-Arbitrage Pricing

The previous examples are simple enough to illustrate the application of constraints in neural networks. However, one would typically need to enforce more complex constraints for no-arbitrage pricing and calibration. Pricing approximations should be monotonically increasing w.r.t. to maturity and convex w.r.t. strike. Such constraints require that the neural network is fitted with more input variables K and T .

Accelerating Calibrations

One promising direction, which does not require neural network derivative pricing, is to simply learn a stochastic volatility based pricing model, such as the Heston model, as a function of underlying price, strike, and maturity, and then use the neural network pricing function to calibrate the pricing model. Such a calibration

Fig. 4.15 The out-of-sample MLP estimation of implied volatility as a function of moneyness is compared with the true values



avoids fitting a few parameters to the chain of observed option prices or implied volatilities. Replacement of expensive pricing functions, which may require FFTs or Monte Carlo methods, with trained neural networks reduces calibration time considerably. See Horvath et al. (2019) for further details.

Dupire Local Volatility

Another challenge is how to price exotic options consistently with the market prices of their European counterpart. The former are typically traded over-the-counter, whereas the latter are often exchange traded and therefore “fully” observable. To fix ideas, let $C(K, T)$ denote an observed call price, for some fixed strike, K , maturity, T , and underlying price S_t . Modulo a short rate and dividend term, the unique “effective” volatility, σ_0^2 , is given by the Dupire formula:

$$\sigma_0^2 = \frac{2\partial_T C(K, T)}{K^2 \partial_K^2 C(K, T)}. \quad (4.34)$$

The challenge arises when calibrating the local volatility model, extracting effective volatility from market option prices is an ill-posed inverse problem. Such a challenge has recently been addressed by Chataigner et al. (2020) in their paper on deep local volatility.

? Multiple Choice Question 2

Which of the following statements are true:

1. A feedforward architecture is always convex w.r.t. each input variables if every activation function is convex and the weights are constrained to be either all positive or all negative.
2. A feedforward architecture with positive weights is a monotonically increasing function of the input for any choice of monotonically increasing activation function.
3. The weights of a feedforward architecture must be constrained for the output of a feedforward network to be bounded.
4. The bias terms in a network simply shift the output and have no effect on the derivatives of the output w.r.t. to the input.

3.1 Similarity of MLPs with Other Supervised Learners

Under special circumstances, MLPs are functionally equivalent to a number of other machine learning techniques. As previously mentioned, when the network has no hidden layer, it is either a regression or logistic regression. Neural networks with

one hidden layer is essentially a projection pursuit regression (PPR), both project the input vector onto a hyperplane, apply a non-linear transformation into feature space, followed by an affine transformation. The mapping of input vectors to feature space by the hidden layer is conceptually similar to kernel methods, such as support vector machines (SVMs), which map to a kernel space, where classification and regression are subsequently performed. Boosted decision stumps, one level boosted decision trees, can even be expressed as a single-layer MLP. Caution must be exercised in over-stretching these conceptual similarities. Data generation assumptions aside, there are differences in the classes of non-linear functions and learning algorithms used. For example, the non-linear function being fitted in PPR can be different for each combination of input variables and is sequentially estimated before updating the weights. In contrast, neural networks fix these functions and estimate all the weights belonging to a single layer simultaneously. A summary of other machine learning approaches is given in Table 4.1 and we refer the reader to numerous excellent textbooks (Bishop 2006; Hastie et al. 2009) covering such methods.

Table 4.1 This table compares supervised machine learning algorithms (reproduced from Mulinainathan and Spiess (2017))

Function class \mathcal{F} (and its parameterization)	Regularizer $R(f)$
<i>Global/parametric predictors</i>	
Linear $\beta'x$ (and generalizations)	Subset selection $\ \beta\ _0 = \sum_{j=1}^k \mathbf{1}_{\beta_j \neq 0}$
	LASSO $\ \beta\ _1 = \sum_{j=1}^k \beta_j $
	Ridge $\ \beta\ _2^2 = \sum_{j=1}^k \beta_j^2$
	Elastic net $\alpha\ \beta\ _1 + (1 - \alpha)\ \beta\ _2^2$
<i>Local/non-parametric predictors</i>	
Decision/regression trees	Depth, number of nodes/leaves, minimal leaf size, information gain at splits
Random forest (linear combination of trees)	Number of trees, number of variables used in each tree, size of bootstrap sample, complexity of trees (see above)
Nearest neighbors	Number of neighbors
Kernel regression	Kernel bandwidth
<i>Mixed predictors</i>	
Deep learning, neural nets, convolutional neural networks	Number of levels, number of neurons per level, connectivity between neurons
Splines	Number of knots, order
<i>Combined predictors</i>	
Bagging: unweighted average of predictors from bootstrap draws	Number of draws, size of bootstrap samples (and individual regularization parameters)
Boosting: linear combination of predictions of residual	Learning rate, number of iterations (and individual regularization parameters)
Ensemble: weighted combination of different predictors	Ensemble weights (and individual regularization parameters)

4 Training, Validation, and Testing

Deep learning is a data-driven approach which focuses on finding structure in large datasets. The main tools for variable or predictor selection are regularization and dropout. Out-of-sample predictive performance helps assess the optimal amount of regularization, the problem of finding the optimal hyperparameter selection. There is still a very Bayesian flavor to the modeling procedure and the modeler follows two key steps:

1. Training phase: pair the input with expected output, until a sufficiently close match has been found. Gauss' original least squares procedure is a common example.
2. Validation and test phase: assess how well the deep learner has been trained for out-of-sample prediction. This depends on the size of your data, the value you would like to predict, the input, etc., and various model properties including the mean-error for numeric predictors and classification errors for classifiers.

Often, the validation phase is split into two parts.

- 2.a First, estimate the out-of-sample accuracy of all approaches (a.k.a. validation).
- 2.b Second, compare the models and select the best performing approach based on the validation data (a.k.a. verification).

Step 2.b. can be skipped if there is no need to select an appropriate model from several rivaling approaches. The researcher then only needs to partition the dataset into a training and test set.

To construct and evaluate a learning machine, we start with training data of input-output pairs $D = \{Y^{(i)}, X^{(i)}\}_{i=1}^N$. The goal is to find the machine learner of $Y = F(X)$, where we have a loss function $\mathcal{L}(Y, \hat{Y})$ for a predictor, \hat{Y} , of the output signal, Y . In many cases, there is an underlying probability model, $p(Y | \hat{Y})$, then the loss function is the negative log probability $\mathcal{L}(Y, \hat{Y}) = -\log p(Y | \hat{Y})$. For example, under a Gaussian model $\mathcal{L}(Y, \hat{Y}) = \|Y - \hat{Y}\|^2$ is a L^2 norm, for binary classification, $\mathcal{L}(Y, \hat{Y}) = -Y \log \hat{Y}$ is the negative cross-entropy. In its simplest form, we then solve an optimization problem

$$\underset{W,b}{\text{minimize}} f(W, b) + \lambda \phi(W, b)$$

$$f(W, b) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(Y^{(i)}, \hat{Y}(X^{(i)}))$$

with a regularization penalty, $\phi(W, b)$. The loss function is non-convex, possessing many local minima and is generally difficult to find a global minimum. An important assumption, which is often not explicitly stated, is that the errors are assumed to be “homoscedastic.” Homoscedasticity is the assumption that the error has an identical distribution over each observation. This assumption can be relaxed by

weighting the observations differently. However, we shall regard such extensions as straightforward and compatible with the algorithms for solving the unweighted optimization problem. Here λ is a global regularization parameter which we tune using the out-of-sample predictive mean squared error (MSE) of the model. The regularization penalty, $\phi(W, b)$, introduces a bias-variance tradeoff. $\nabla \mathcal{L}$ is given in closed form by a chain rule and, through back-propagation, each layer's weights $\hat{W}^{(\ell)}$ are fitted with stochastic gradient descent.

Recall from Chap. 1 that a 1-of- K encoding is used for a categorical response, so that G is a K -binary vector $G \in [0, 1]^K$ and the value k is presented as $G_k = 1$ and $G_j = 0, \forall j \neq k$, where $\|G\|_1 = 1$. The predictor is given by $\hat{G}_k := g_k(X|(W, b))$, $\|\hat{G}\|_1 = 1$ and the loss function is the negative cross-entropy for discrete random variables

$$\mathcal{L}(G, \hat{G}(X)) = -G^T \ln \hat{G}. \quad (4.35)$$

For example, if there are $K = 3$ classes, then $G = [0, 0, 1]$, $G = [0, 1, 0]$, or $G = [1, 0, 0]$ to represent the three classes. When $K > 2$, the output layer has K neurons and the loss function is the negative cross-entropy

$$\mathcal{L}(G, \hat{G}(X)) = - \sum_{k=1}^K G_k \ln \hat{G}_k. \quad (4.36)$$

For the case when $K = 2$, i.e. binary classification, there is only one neuron in the output layer and the loss function is

$$\mathcal{L}(G, \hat{G}(X)) = -G \ln \hat{G} - (1 - G) \ln (1 - \hat{G}), \quad (4.37)$$

where $\hat{G} = g_1(X|(W, b)) = \sigma(I^{(L-1)})$ and σ is a sigmoid function.

We observe that when there are no hidden layers, $I^{(1)} = W^{(1)}X + b^{(1)}$ and $g_1(X|(W, b))$ is a logistic regression. The *softmax* function, σ_s generalizes binary classifiers to multi-classifiers. $\sigma_s : \mathbb{R}^K \rightarrow [0, 1]^K$ is a continuous K -vector function given by

$$\sigma_s(\mathbf{x})_k = \frac{\exp(x_k)}{\|\exp(\mathbf{x})\|_1}, \quad k \in \{1, \dots, K\}, \quad (4.38)$$

where $\|\sigma_s(\mathbf{x})\|_1 = 1$. The *softmax* function is used to represent a probability distribution over K possible states:

$$\hat{G}_k = P(G = k | X) = \sigma_s(WX + b) = \frac{\exp((WX + b)_k)}{\|\exp(WX + b)\|_1}. \quad (4.39)$$

► Derivative of the Softmax Function

Using the quotient rule $f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{[h(x)]^2}$, the derivative $\sigma := \sigma_s(\mathbf{x})$ can be written as:

$$\frac{\partial \sigma_i}{\partial x_i} = \frac{\exp(x_i) \|\exp(\mathbf{x})\|_1 - \exp(x_i) \exp(x_i)}{\|\exp(\mathbf{x})\|_1^2} \quad (4.40)$$

$$= \frac{\exp(x_i)}{\|\exp(\mathbf{x})\|_1} \cdot \frac{\|\exp(\mathbf{x})\|_1 - \exp(x_i)}{\|\exp(\mathbf{x})\|_1} \quad (4.41)$$

$$= \sigma_i(1 - \sigma_i) \quad (4.42)$$

For the case $i \neq j$, the derivative of the sum is

$$\frac{\partial \sigma_i}{\partial x_j} = \frac{0 - \exp(x_i) \exp(x_j)}{\|\exp(\mathbf{x})\|_1^2} \quad (4.43)$$

$$= -\frac{\exp(x_j)}{\|\exp(\mathbf{x})\|_1} \cdot \frac{\exp(x_i)}{\|\exp(\mathbf{x})\|_1} \quad (4.44)$$

$$= -\sigma_j \sigma_i \quad (4.45)$$

This can be written compactly as $\frac{\partial \sigma_i}{\partial x_j} = \sigma_i(\delta_{ij} - \sigma_j)$, where δ_{ij} is the Kronecker delta function.

5 Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) method or its variation is typically used to find the deep learning model weights by minimizing the penalized loss function, $f(W, b)$. The method minimizes the function by taking a negative step along an estimate g^k of the gradient $\nabla f(W^k, b^k)$ at iteration k . The approximate gradient is calculated by

$$g^k = \frac{1}{b_k} \sum_{i \in E_k} \nabla \mathcal{L}_{W,b}(Y^{(i)}, \hat{Y}^k(X^{(i)})),$$

where $E_k \subset \{1, \dots, N\}$ and $b_k = |E_k|$ is the number of elements in E_k (a.k.a. batch size). When $b_k > 1$ the algorithm is called batch SGD and simply SGD otherwise. A usual strategy to choose subset E is to go cyclically and pick consecutive elements of $\{1, \dots, N\}$ and $E_{k+1} = [E_k \bmod N] + 1$, where modular arithmetic is applied to the set. The approximated direction g^k is calculated using a chain rule (a.k.a. back-propagation) for deep learning. It is an unbiased estimator of $\nabla f(W^k, b^k)$, and we have

$$\mathbb{E}(g^k) = \frac{1}{N} \sum_{i=1}^N \nabla \mathcal{L}_{W,b} \left(Y^{(i)}, \hat{Y}^k(X^{(i)}) \right) = \nabla f(W^k, b^k).$$

At each iteration, we update the solution $(W, b)^{k+1} = (W, b)^k - t_k g^k$.

Deep learning applications use a step size t_k (a.k.a. learning rate) as constant or a reduction strategy of the form, $t_k = a \exp\{-kt\}$. Appropriate learning rates or the hyperparameters of reduction schedule are usually found empirically from numerical experiments and observations of the loss function progression. In order to update the weights across the layers, *back-propagation* is needed and will now be explained.

? Multiple Choice Question 3

Which of the following statements are true:

1. The training of a neural network involves minimizing a loss function w.r.t. the weights and biases over the training data.
 2. L_1 regularization is used during model selection to penalize models with too many parameters.
 3. In deep learning, regularization can be applied to each layer of the network.
 4. Back-propagation uses the chain rule to update the weights of the network but is not guaranteed to converge to a unique minimum.
 5. Stochastic gradient descent and back-propagation are two different optimization algorithms for minimizing the loss function and the user must choose the best one.
-

5.1 Back-Propagation

Staying with a multi-classifier, we can begin by informally motivating the need for a recursive approach to updating the weights and biases. Let us express $\hat{Y} \in [0, 1]^K$ as a function of the final weight matrix $W \in \mathbb{R}^{K \times M}$ and output bias \mathbb{R}^K so that

$$\hat{Y}(W, b) = \sigma \circ I(W, b), \quad (4.46)$$

where the input function $I : \mathbb{R}^{K \times M} \times \mathbb{R}^K \rightarrow \mathbb{R}^K$ is of the form $I(W, b) := WX + b$ and $\sigma : \mathbb{R}^K \rightarrow \mathbb{R}^K$ is the softmax function. Applying the multivariate chain rule gives the Jacobian of $\hat{Y}(W, b)$:

$$\nabla \hat{Y}(W, b) = \nabla(\sigma \circ I)(W, b) \quad (4.47)$$

$$= \nabla \sigma(I(W, b)) \cdot \nabla I(W, b). \quad (4.48)$$

5.1.1 Updating the Weight Matrices

Recall that the loss function for a multi-classifier is the cross-entropy

$$\mathcal{L}(Y, \hat{Y}(X)) = - \sum_{k=1}^K Y_k \ln \hat{Y}_k. \quad (4.49)$$

Since Y is a constant vector we can express the cross-entropy as a function of (W, b)

$$\mathcal{L}(W, b) = \mathcal{L} \circ \sigma(I(W, b)). \quad (4.50)$$

Applying the multivariate chain rule gives

$$\nabla \mathcal{L}(W, b) = \nabla(\mathcal{L} \circ \sigma)(I(W, b)) \quad (4.51)$$

$$= \nabla \mathcal{L}(\sigma(I(W, b))) \cdot \nabla \sigma(I(W, b)) \cdot \nabla I(W, b). \quad (4.52)$$

Stochastic gradient descent is used to find the minimum

$$(\hat{W}, \hat{b}) = \arg \min_{W, b} \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y_i, \hat{Y}^{W, b}(\mathbf{x}_i)). \quad (4.53)$$

Because of the compositional form of the model, the gradient must be derived using the chain rule for differentiation. This can be computed by a forward and then a backward sweep (“back-propagation”) over the network, keeping track only of quantities local to each neuron.

Forward Pass

Set $Z^{(0)} = X$ and for $\ell \in \{1, \dots, L\}$ set

$$Z^{(\ell)} = f_{W^{(\ell)}, b^{(\ell)}}^{(\ell)}(Z^{(\ell-1)}) = \sigma^{(\ell)}(W^{(\ell)} Z^{(\ell-1)} + b^{(\ell)}). \quad (4.54)$$

On completion of the forward pass, the error $\hat{Y} - Y$ is evaluated using $\hat{Y} := Z^{(L)}$.

Back-Propagation

Define the back-propagation error $\delta^{(\ell)} := \nabla_{b^{(\ell)}} \mathcal{L}$, where given $\delta^{(L)} = \hat{Y} - Y$, and for $\ell = L-1, \dots, 1$ the following recursion relation gives the updated back-propagation error and weight update for layer ℓ :

$$\delta^{(\ell)} = (\nabla_{I^{(\ell)}} \sigma^{(\ell)}) W^{(\ell+1)T} \delta^{(\ell+1)}, \quad (4.55)$$

$$\nabla_{W^{(\ell)}} \mathcal{L} = \delta^{(\ell)} \otimes Z^{(\ell-1)}, \quad (4.56)$$

and \otimes is the outer product of two vectors. See Appendix “Back-Propagation” for a derivation of Eqs. 4.55 and 4.56.

The weights and biases are updated for all $\ell \in \{1, \dots, L\}$ according to the expression

$$\begin{aligned} \Delta W^{(\ell)} &= -\gamma \nabla_{W^{(\ell)}} \mathcal{L} = -\gamma \delta^{(\ell)} \otimes Z^{(\ell-1)}, \\ \Delta b^{(\ell)} &= -\gamma \delta^{(\ell)}, \end{aligned}$$

where γ is a user defined learning rate parameter. Note the negative sign: this indicates that weight changes are in the direction of decrease in error. *Mini-batch* or off-line updates involve using many observations of X at the same time. The *batch size* refers to the number of observations of X used in each pass. An *epoch* refers to a round-trip (i.e., forward+backward pass) over all training samples.

Example 4.5 Back-Propagation with a Three-Layer Network

Suppose that a feedforward network classifier has two sigmoid activated hidden layers and a softmax activated output layer. After a forward pass, the values of $\{Z^{(\ell)}\}_{\ell=1}^3$ are stored and the error $\hat{Y} - Y$, where $\hat{Y} = Z^{(3)}$, is calculated for one observation of X . The back-propagation and weight updates in the final layer are evaluated:

$$\begin{aligned} \delta^{(3)} &= \hat{Y} - Y \\ \nabla_{W^{(3)}} \mathcal{L} &= \delta^{(3)} \otimes Z^{(2)}. \end{aligned}$$

Now using Eqs. 4.55 and 4.56, we update the back-propagation error and weight updates for hidden layer 2

$$\begin{aligned} \delta^{(2)} &= Z^{(2)}(1 - Z^{(2)})(W^{(3)})^T \delta^{(3)}, \\ \nabla_{W^{(2)}} \mathcal{L} &= \delta^{(2)} \otimes Z^{(1)}. \end{aligned}$$

(continued)

Example 4.4 (continued)

Repeating for hidden layer 1

$$\delta^{(1)} = Z^{(1)}(1 - Z^{(1)})(W^{(2)})^T \delta^{(2)},$$

$$\nabla_{W^{(1)}} \mathcal{L} = \delta^{(1)} \otimes X.$$

We update the weights and biases using Eqs. 4.57 and 4.57, so that $b^{(3)} \rightarrow b^{(3)} - \gamma \delta^{(3)}$, $W^{(3)} \rightarrow W^{(3)} - \gamma \delta^{(3)} \otimes Z^{(2)}$ and repeat for the other weight-bias pairs, $\{(W^{(\ell)}, b^{(\ell)})\}_{\ell=1}^2$. See the back-propagation notebook for further details of a worked example in Python and then complete Exercise 4.12.

5.2 Momentum

One disadvantage of SGD is that the descent in f is not guaranteed or can be very slow at every iteration. Furthermore, the variance of the gradient estimate g^k is near zero as the iterates converge to a solution. To address those problems a coordinate descent (CD) and momentum-based modifications of SGD are used. Each CD step evaluates a single component E_k of the gradient ∇f at the current point and then updates the E_k th component of the variable vector in the negative gradient direction. The momentum-based versions of SGD or the so-called accelerated algorithms were originally proposed by Nesterov (2013).

The use of momentum in the choice of step in the search direction combines new gradient information with the previous search direction. These methods are also related to other classical techniques such as the heavy-ball method and conjugate gradient methods. Empirically momentum-based methods show a far better convergence for deep learning networks. The key idea is that the gradient only influences changes in the “velocity” of the update

$$v^{k+1} = \mu v^k - t_k g^k,$$

$$(W, b)^{k+1} = (W, b)^k + v^k.$$

The parameter μ controls the dumping effect on the rate of update of the variables. The physical analogy is the reduction in kinetic energy that allows “slow down” in the movements at the minima. This parameter is also chosen empirically using cross-validation.

Nesterov’s momentum method (a.k.a. Nesterov acceleration) instead calculate gradient at the point predicted by the momentum. We can think of it as a look-ahead strategy. The resulting update equations are

$$\begin{aligned} v^{k+1} &= \mu v^k - t_k g((W, b)^k + v^k), \\ (W, b)^{k+1} &= (W, b)^k + v^k. \end{aligned}$$

Another popular modification to the SGD method is the AdaGrad method, which adaptively scales each of the learning parameters at each iteration

$$\begin{aligned} c^{k+1} &= c^k + g((W, b)^k)^2, \\ (W, b)^{k+1} &= (W, b)^k - t_k g(W, b)^k / (\sqrt{c^{k+1}} - a), \end{aligned}$$

where a is usually a small number, e.g. $a = 10^{-6}$ that prevents dividing by zero. PRMSprop takes the AdaGrad idea further and places more weight on recent values of the gradient squared to scale the update direction, i.e. we have

$$c^{k+1} = dc^k + (1-d)g((W, b)^k)^2.$$

The Adam method combines both PRMSprop and momentum methods and leads to the following update equations:

$$\begin{aligned} v^{k+1} &= \mu v^k - (1-\mu)t_k g((W, b)^k + v^k), \\ c^{k+1} &= dc^k + (1-d)g((W, b)^k)^2, \\ (W, b)^{k+1} &= (W, b)^k - t_k v^{k+1} / (\sqrt{c^{k+1}} - a). \end{aligned}$$

Second-order methods solve the optimization problem by solving a system of non-linear equations $\nabla f(W, b) = 0$ with Newton's method

$$(W, b)^+ = (W, b) - \{\nabla^2 f(W, b)\}^{-1} \nabla f(W, b).$$

SGD simply approximates $\nabla^2 f(W, b)$ by $1/t$. The advantages of a second-order method include much faster convergence rates and insensitivity to the conditioning of the problem. In practice, second-order methods are rarely used for deep learning applications (Dean et al. 2012). The major disadvantage is the inability to train the model using batches of data as SGD does. Since typical deep learning models rely on large-scale datasets, second-order methods become memory and computationally prohibitive at even modest-sized training datasets.

5.2.1 Computational Considerations

Batching alone is not sufficient to scale SGD methods to large-scale problems on modern high-performance computers. Back-propagation through a chain rule creates an inherit sequential dependency in the weight updates which limits the

dataset dimensions for the deep learner. Polson et al. (2015) consider a proximal Newton method, a Bayesian optimization technique which provides an efficient solution for estimation and optimization of such models and for calculating a regularization path. The authors present a splitting approach, alternating direction method of multipliers (ADMM), which overcomes the inherent bottlenecks in back-propagation by providing a simultaneous block update of parameters at all layers. ADMM facilitates the use of large-scale computing.

A significant factor in the widespread adoption of deep learning has been the creation of TensorFlow (Abadi et al. 2016), an interface for easily expressing machine learning algorithms and mapping compute intensive operations onto a wide variety of different hardware platforms and in particular GPU cards. Recently, TensorFlow has been augmented by Edward (Tran et al. 2017) to combine concepts in Bayesian statistics and probabilistic programming with deep learning.

5.2.2 Model Averaging via Dropout

We close this section by briefly mentioning one final technique which has proved indispensable in preventing neural networks from over-fitting. Dropout is a computationally efficient technique to reduce model variance by considering many model configurations and then averaging the predictions. The layer input space $Z = (Z_1, \dots, Z_n)$, where n is large, needs dimension reduction techniques which are designed to avoid over-fitting in the training process. Dropout works by removing layer inputs randomly with a given probability θ . The probability, θ , can be viewed as a further hyperparameter (like λ) which can be tuned via cross-validation. Heuristically, if there are 1000 variables, then a choice of $\theta = 0.1$ will result in a search for models with 100 variables. The dropout architecture with stochastic search for the predictors can be used

$$\begin{aligned} d_i^{(\ell)} &\sim \text{Ber}(\theta), \\ \tilde{Z}^{(\ell)} &= d^{(\ell)} \circ Z^{(\ell)}, \quad 1 \leq \ell < L, \\ Z^{(\ell)} &= \sigma^{(\ell)}(W^{(\ell)}\tilde{Z}^{(\ell-1)} + b^{(\ell)}). \end{aligned}$$

Effectively, this replaces the layer input Z by $d \circ Z$, where \circ denotes the element-wise product and d is a vector of independent Bernoulli, $\text{Ber}(\theta)$, distributed random variables. The overall objective function is closely related to ridge regression with a g-prior (Heaton et al. 2017).

6 Bayesian Neural Networks*

Bayesian deep learning (Neal 1990; Saul et al. 1996; Frey and Hinton 1999; Lawrence 2005; Adams et al. 2010; Mnih and Gregor 2014; Kingma and Welling 2013; Rezende et al. 2014) provides a powerful and general framework for statistical modeling. Such a framework allows for a completely new approach to data modeling and solves a number of problems that conventional models cannot address: (i) DLs (deep learners) permit complex dependencies between variables to be explicitly represented which are difficult, if not impossible, to model with copulas; (ii) they capture correlations between variables in high-dimensional datasets; and (iii) they characterize the degree of uncertainty in predicting large-scale effects from large datasets relevant for quantifying uncertainty.

Uncertainty refers to the statistically unmeasurable situation of Knightian uncertainty, where the event space is known but the probabilities are not (Chen et al. 2017). Oftentimes, a forecast may be shrouded in uncertainty arising from noisy data or model uncertainty, either through incorrect modeling assumptions or parameter error. It is desirable to characterize this uncertainty in the forecast. In conventional Bayesian modeling, uncertainty is used to learn from small amounts of low-dimensional data under parametric assumptions on the prior. The choice of the prior is typically the point of contention and chosen for solution tractability rather than modeling fidelity. Recently, deterministic deep learners have been shown to scale well to large, high-dimensional, datasets. However, the probability vector obtained from the network is often erroneously interpreted as model confidence (Gal 2016).

A typical approach to model uncertainty in neural network models is to assume that model parameters (weights and biases) are random variables (as illustrated in Fig. 4.16). Then ANN model approaches Gaussian process as the number of weights goes to infinity (Neal 2012; Williams 1997). In the case of finite number of weights, a network with random parameters is called a Bayesian neural network (MacKay 1992b). Recent advances in “variational inference” techniques and software that represent mathematical models as a computational graph (Blundell et al. 2015a) enable probabilistic deep learning models to be built, without having to worry about how to perform testing (forward propagation) or inference (gradient-based optimization, with back-propagation and automatic differentiation). *Variational inference* is an approximate technique which allows multi-modal likelihood functions to be extremized with standard stochastic gradient descent algorithm. An alternative to variational and MCMC algorithms was recently proposed by Gal (2016) and builds on efficient dropout regularization technique.

All of the current techniques rely on approximating the true posterior over the model parameters $p(w | X, Y)$ by another distribution $q_\theta(w)$ which can be evaluated in a computationally tractable way. Such a distribution is chosen to be as close as possible to the true posterior and is found by minimizing the Kullback–Leibler (KL) divergence

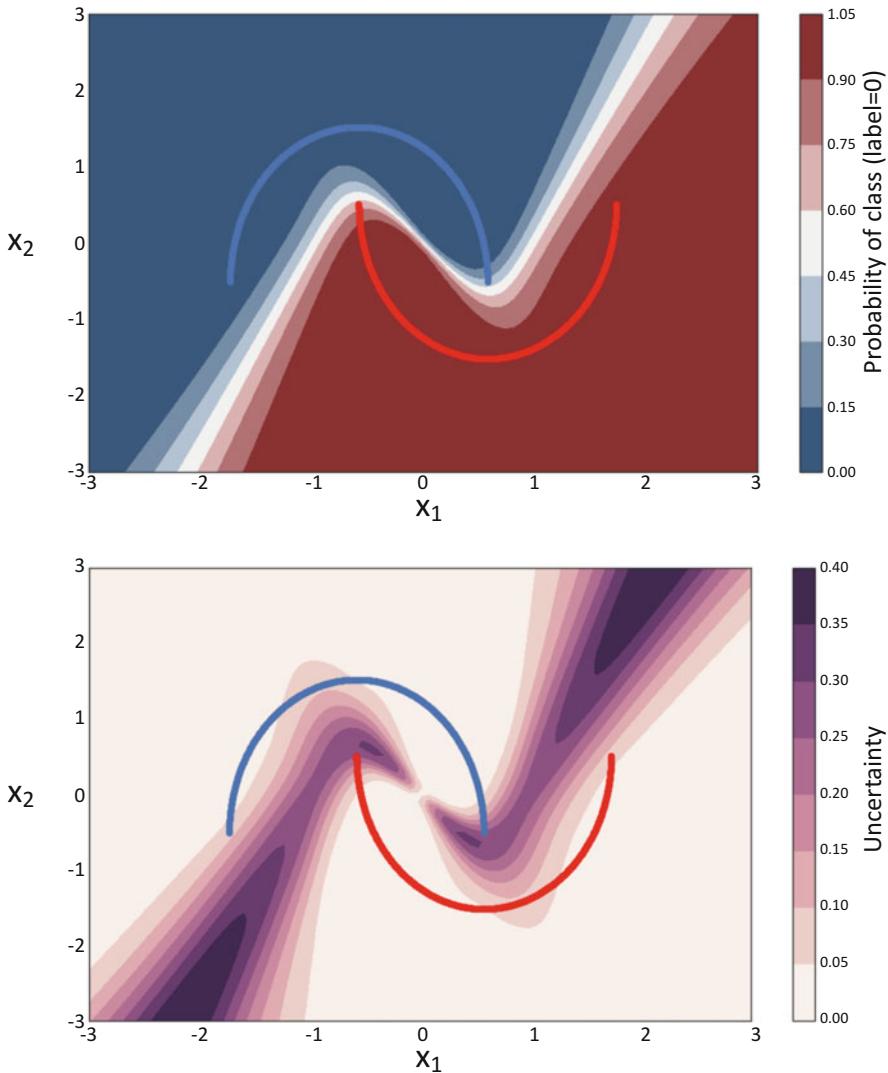


Fig. 4.16 Bayesian classification of the half-moon problem with neural networks. (top) The posterior mean and (bottom) the posterior std. dev.

$$\theta^* \in \arg \min_{\theta} \int q_{\theta}(w) \log \frac{q_{\theta}(w)}{p(w | X, Y)} dw.$$

There are numerous approaches to Bayesian deep learning for uncertainty quantification including MCMC (Markov chain Monte Carlo) methods. These are known to scale poorly with the number of observations and recent studies have

developed SG-MCMC (stochastic gradient MCMC) and related methods such as PX-MCMC (parameter expansion MCMC) to ease the computational burden. A Bayesian extension of feedforward network architectures has been considered by several authors (Neal 1990; Saul et al. 1996; Frey and Hinton 1999; Lawrence 2005; Adams et al. 2010; Mnih and Gregor 2014; Kingma and Welling 2013; Rezende et al. 2014). Recent results show how dropout regularization can be used to represent uncertainty in deep learning models. In particular, Gal (2015) shows that dropout provides uncertainty estimates for the predicted values. The predictions generated by the deep learning models with dropout are nothing but samples from the predictive posterior distribution.

A classical example of using neural networks to model a vector of binary variables is the Boltzmann machine (BM), with two layers. The first layer encodes latent variables and the second layer encodes the observed variables. Both conditional distributions $p(\text{data} \mid \text{latent variables})$ and $p(\text{latent variables} \mid \text{data})$ are specified using logistic functions parameterized by weights and offset vectors. The size of the joint distribution table grows exponentially with the number of variables and (Hinton and Sejnowski 1983) proposed using Gibbs sampler to calculate updates to model weights on each iteration. The multi-modal nature of the posterior distribution leads to prohibitive computational times required to learn models of a practical size. Tieleman (2008) proposed a variational approach that replaces the posterior $p(\text{latent variables} \mid \text{data})$ and approximates it with another easy to calculate distribution and was considered in Salakhutdinov (2008). Several extensions to the BMs have been proposed. Exponential family extensions have been considered by (Smolensky 1986; Salakhutdinov 2008; Salakhutdinov and Hinton 2009; Welling et al. 2005).

There have also been multiple approaches to building inference algorithms for deep learning models (MacKay 1992a; Hinton and Van Camp 1993; Neal 1992; Barber and Bishop 1998). Performing Bayesian inference on a neural network calculates the posterior distribution over the weights given the observations. In general, such a posterior cannot be calculated analytically, or even efficiently sampled from. However, several recently proposed approaches address the computational problem for some specific deep learning models (Graves 2011; Kingma and Welling 2013; Rezende et al. 2014; Blundell et al. 2015b; Hernández-Lobato and Adams 2015; Gal and Ghahramani 2016).

The recent successful approaches to develop efficient Bayesian inference algorithms for deep learning networks are based on the reparameterization techniques for calculating Monte Carlo gradients while performing variational inference. Such an approach has led to an explosive development in the application of stochastic variational inference. Given the data $\mathcal{D} = (X, Y)$, the variational inference relies on approximating the posterior $p(\theta \mid \mathcal{D})$ with a variation distribution $q(\theta \mid \mathcal{D}, \phi)$, where $\theta = (W, b)$. Then q is found by minimizing the Kullback–Leibler divergence between the approximate distribution and the posterior, namely

$$\text{KL}(q \parallel p) = \int q(\theta \mid \mathcal{D}, \phi) \log \frac{q(\theta \mid \mathcal{D}, \phi)}{p(\theta \mid \mathcal{D})} d\theta.$$

Since $p(\theta | \mathcal{D})$ is not necessarily tractable, we replace minimization of $\text{KL}(q || p)$ with maximization of the evidence lower bound (ELBO)

$$\text{ELBO}(\phi) = \int q(\theta | \mathcal{D}, \phi) \log \frac{p(Y | X, \theta) p(\theta)}{q(\theta | \mathcal{D}, \phi)} d\theta$$

The *log* of the total probability (evidence) is then

$$\log p(D) = \text{ELBO}(\phi) + \text{KL}(q || p).$$

The sum does not depend on ϕ , thus minimizing $\text{KL}(q || p)$ is the same as maximizing $\text{ELBO}(\phi)$. Also, since $\text{KL}(q || p) \geq 0$, which follows from Jensen's inequality, we have $\log p(\mathcal{D}) \geq \text{ELBO}(\phi)$. Thus, the evidence lower bound name. The resulting maximization problem $\text{ELBO}(\phi) \rightarrow \max_{\phi}$ is solved using stochastic gradient descent.

To calculate the gradient, it is convenient to write the ELBO as

$$\text{ELBO}(\phi) = \int q(\theta | \mathcal{D}, \phi) \log p(Y | X, \theta) d\theta - \int q(\theta | \mathcal{D}, \phi) \log \frac{q(\theta | \mathcal{D}, \phi)}{p(\theta)} d\theta$$

The gradient of the first term $\nabla_{\phi} \int q(\theta | \mathcal{D}, \phi) \log p(Y | X, \theta) d\theta = \nabla_{\phi} E_q \log p(Y | X, \theta)$ is not an expectation and thus cannot be calculated using Monte Carlo methods. The idea is to represent the gradient $\nabla_{\phi} E_q \log p(Y | X, \theta)$ as an expectation of some random variable, so that Monte Carlo techniques can be used to calculate it. There are two standard methods to do it. First, the log-derivative trick uses the following identity $\nabla_x f(x) = f(x) \nabla_x \log f(x)$ to obtain $\nabla_{\phi} E_q \log p(Y | \theta)$. Thus, if we select $q(\theta | \phi)$ so that it is easy to compute its derivative and generate samples from it, the gradient can be efficiently calculated using Monte Carlo techniques. Second, we can use the reparameterization trick by representing θ as a value of a deterministic function, $\theta = g(\epsilon, x, \phi)$, where $\epsilon \sim r(\epsilon)$ does not depend on ϕ . The derivative is given by

$$\begin{aligned} \nabla_{\phi} E_q \log p(Y | X, \theta) &= \int r(\epsilon) \nabla_{\phi} \log p(Y | X, g(\epsilon, x, \phi)) d\epsilon \\ &= E_{\epsilon} [\nabla_g \log p(Y | X, g(\epsilon, x, \phi)) \nabla_{\phi} g(\epsilon, x, \phi)]. \end{aligned}$$

The reparameterization is trivial when $q(\theta | \mathcal{D}, \phi) = \mathcal{N}(\theta | \mu(\mathcal{D}, \phi), \Sigma(\mathcal{D}, \phi))$, and $\theta = \mu(\mathcal{D}, \phi) + \epsilon \Sigma(\mathcal{D}, \phi)$, $\epsilon \sim \mathcal{N}(0, I)$. Kingma and Welling (2013) propose using $\Sigma(\mathcal{D}, \phi) = I$ and representing $\mu(\mathcal{D}, \phi)$ and ϵ as outputs of a neural network (multilayer perceptron), the resulting approach was called a variational autoencoder. A generalized reparameterization has been proposed by Ruiz et al. (2016) and combines both log-derivative and reparameterization techniques by assuming that ϵ can depend on ϕ .

7 Summary

In this chapter we have introduced some of the theory of function approximation and out-of-sample estimation with neural networks when the observation points are i.i.d. Such a case is not suitable for times series data and shall be the subject of later chapters. We restricted our attention to feedforward neural networks in order to explore some of the theoretical arguments which help us reason scientifically about architecture design. We have seen that feedforward networks use hidden units, or perceptrons, to partition the input space into regions bounded with manifolds. In the case of ReLU activated units, each manifold is a hyperplane and the hidden units form a hyperplane arrangement. We have introduced various approaches to reason about the effect of the number of units in each layer in addition to reasoning about the effect of hidden layers. We also introduced various concepts and methods necessary for understanding and applying neural networks to i.i.d. data including

- Fat shattering, VC dimension, and the empirical risk measure (ERM) as the basis for characterizing the learnability of a class of MLPs;
- The construction of neural networks as splines and their pointwise approximation error bound;
- The reason for composing layers in deep learning;
- Stochastic gradient descent and back-propagation as techniques for training neural networks; and
- Imposing constraints on the network needed for approximating financial derivatives and other constrained optimization problems in finance.

8 Exercises

Exercise 4.1

Show that substituting

$$\nabla_{ij} I_k = \begin{cases} X_j, & i = k, \\ 0, & i \neq k, \end{cases}$$

into Eq. 4.47 gives

$$\nabla_{ij} \sigma_k \equiv \frac{\partial \sigma_k}{\partial w_{ij}} = \nabla_i \sigma_k X_j = \sigma_k (\delta_{ki} - \sigma_i) X_j.$$

Exercise 4.2

Show that substituting the derivative of the softmax function w.r.t. w_{ij} into Eq. 4.52 gives for the special case when the output is $Y_k = 1$, $k = i$, and $Y_k = 0$, $\forall k \neq i$:

$$\nabla_{ij} \mathcal{L}(W, b) := [\nabla_W \mathcal{L}(W, b)]_{ij} = \begin{cases} (\sigma_i - 1)X_j, & Y_i = 1, \\ 0, & Y_k = 0, \forall k \neq i. \end{cases}$$

Exercise 4.3

Consider feedforward neural networks constructed using the following two types of activation functions:

- Identity

$$Id(x) := x$$

- Step function (a.k.a. Heaviside function)

$$H(x) := \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

1. Consider a feedforward neural network with one input $x \in \mathbb{R}$, a single hidden layer with K units having step function activations, $H(x)$, and a single output with identity (a.k.a. linear) activation, $Id(x)$. The output can be written as

$$\hat{f}(x) = Id \left(b^{(2)} + \sum_{k=1}^K w_k^{(2)} H(b_k^{(1)} + w_k^{(1)} x) \right).$$

Construct neural networks using these activation functions.

- a. Consider the step function

$$u(x; a) := yH(x - a) = \begin{cases} y, & \text{if } x \geq a, \\ 0, & \text{otherwise.} \end{cases}$$

Construct a neural network with one input x and one hidden layer, whose response is $u(x; a)$. Draw the structure of the neural network, specify the activation function for each unit (either Id or H), and specify the values for all weights (in terms of a and y).

- b. Now consider the indicator function

$$\mathbf{1}_{[a,b)}(x) = \begin{cases} 1, & \text{if } x \in [a, b), \\ 0, & \text{otherwise.} \end{cases}$$

Construct a neural network with one input x and one hidden layer, whose response is $y\mathbf{1}_{[a,b)}(x)$, for given real values y, a and b . Draw the structure of the neural network, specify the activation function for each unit (either Id or H), and specify the values for all weights (in terms of a, b and y).

Exercise 4.4

A neural network with a single hidden layer can provide an arbitrarily close approximation to any 1-dimensional bounded smooth function. This question will guide you through the proof. Let $f(x)$ be any function whose domain is $[C, D]$, for real values $C < D$. Suppose that the function is Lipschitz continuous, that is,

$$\forall x, x' \in [C, D], |f(x') - f(x)| \leq L|x' - x|,$$

for some constant $L \geq 0$. Use the building blocks constructed in the previous part to construct a neural network with one hidden layer that approximates this function within $\epsilon > 0$, that is, $\forall x \in [C, D], |f(x) - \hat{f}(x)| \leq \epsilon$, where $\hat{f}(x)$ is the output of your neural network given input x . Your network should use only the identity or the Heaviside activation functions. You need to specify the number K of hidden units, the activation function for each unit, and a formula for calculating each weight w_0 , w_k , $w_0^{(k)}$, and $w_1^{(k)}$, for each $k \in \{1, \dots, K\}$. These weights may be specified in terms of C, D, L , and ϵ , as well as the values of $f(x)$ evaluated at a finite number of x values of your choosing (you need to explicitly specify which x values you use). You do not need to explicitly write the $\hat{f}(x)$ function. Why does your network attain the given accuracy ϵ ?

Exercise 4.5

Consider a shallow neural network regression model with n tanh activated units in the hidden layer and d outputs. The hidden-outer weight matrix $W_{ij}^{(2)} = \frac{1}{n}$ and the input-hidden weight matrix $W^{(1)} = 1$. The biases are zero. If the features, X_1, \dots, X_p are i.i.d. Gaussian random variables with mean $\mu = 0$, variance σ^2 , show that

- a. $\hat{Y} \in [-1, 1]$.
- b. \hat{Y} is independent of the number of hidden units, $n \geq 1$.
- c. The expectation, $\mathbb{E}[\hat{Y}] = 0$, and the variance $\mathbb{V}[\hat{Y}] \leq 1$.

Exercise 4.6

Determine the VC dimension of the sum of indicator functions where $\Omega = [0, 1]$

$$F_k(x) = \{f : \Omega \rightarrow \{0, 1\}, f(x) = \sum_{i=0}^k \mathbf{1}_{x \in [t_{2i}, t_{2i+1})}, 0 \leq t_0 < \dots < t_{2k+1} \leq 1, k \geq 1\}.$$

Exercise 4.7

Show that a feedforward binary classifier with two Heaviside activated units shatters the data $\{0.25, 0.5, 0.75\}$.

Exercise 4.8

Compute the weight and bias updates of $W^{(2)}$ and $b^{(2)}$ given a shallow binary classifier (with one hidden layer) with unit weights, zero biases, and ReLU activation of two hidden units for the labeled observation $(x = 1, y = 1)$.

8.1 Programming Related Questions*

Exercise 4.9

Consider the following dataset (taken from Anscombe's quartet):

$$(x_1, y_1) = (10.0, 9.14), (x_2, y_2) = (8.0, 8.14), (x_3, y_3) = (13.0, 8.74), \\ (x_4, y_4) = (9.0, 8.77), (x_5, y_5) = (11.0, 9.26), (x_6, y_6) = (14.0, 8.10), \\ (x_7, y_7) = (6.0, 6.13), (x_8, y_8) = (4.0, 3.10), (x_9, y_9) = (12.0, 9.13), \\ (x_{10}, y_{10}) = (7.0, 7.26), (x_{11}, y_{11}) = (5.0, 4.74).$$

- a. Use a neural network library of your choice to show that a feedforward network with one hidden layer consisting of one unit and a feedforward network with no hidden layers, each using only linear activation functions, do not outperform linear regression based on ordinary least squares (OLS).
- b. Also demonstrate that a neural network with a hidden layer of three neurons using the tanh activation function and an output layer using the linear activation function captures the non-linearity and outperforms the linear regression.

Exercise 4.10

Review the Python notebook `deep_classifiers.ipynb`. This notebook uses Keras to build three simple feedforward networks applied to the half-moon problem: a logistic regression (with no hidden layer); a feedforward network with one hidden layer; and a feedforward architecture with two hidden layers. The half-moons problem is not linearly separable in the original coordinates. However you will observe—after plotting the fitted weights and biases—that a network with many hidden neurons gives a linearly separable representation of the classification problem in the coordinates of the output from the final hidden layer.

Complete the following questions in your own words.

- a. Did we need more than one hidden layer to perfectly classify the half-moons dataset? If not, why might multiple hidden layers be useful for other datasets?
- b. Why not use a very large number of neurons since it is clear that the classification accuracy improves with more degrees of freedom?
- c. Repeat the plotting of the hyperplane, in Part 1b of the notebook, only without the ReLU function (i.e., `activation="linear"`). Describe qualitatively how the decision surface changes with increasing neurons. Why is a (non-linear) activation function needed? The use of figures to support your answer is expected.

Exercise 4.11

Using the `EarlyStopping` callback in Keras, modify the notebook `Deep_Classifiers.ipynb` to terminate training under the following stopping criterion $|L^{(k+1)} - L^{(k)}| \leq \delta$ with $\delta = 0.1$.

Exercise 4.12***

Consider a feedforward neural network with three inputs, two units in the first hidden layer, two units in the second hidden layer, and three units in the output layer. The activation function for hidden layer 1 is ReLU, for hidden layer 2 is sigmoid, and for the output layer is softmax.

The initial weights are given by the matrices

$$W^{(1)} = \begin{pmatrix} 0.1 & 0.3 & 0.7 \\ 0.9 & 0.4 & 0.4 \end{pmatrix}, W^{(2)} = \begin{pmatrix} 0.4 & 0.3 \\ 0.7 & 0.2 \end{pmatrix}, W^{(3)} = \begin{pmatrix} 0.5 & 0.6 \\ 0.6 & 0.7 \\ 0.3 & 0.2 \end{pmatrix},$$

and all the biases are unit vectors.

Assuming that the input $(0.1 \ 0.7 \ 0.3)$ corresponds to the output $(1 \ 0 \ 0)$, manually compute the updated weights and biases after a single epoch (forward + backward pass), clearly stating all derivatives that you have used. You should use a learning rate of 1.

As a practical exercise, you should modify the implementation of a stochastic gradient descent routine in the back-propagation Python notebook.

Note that the notebook example corresponds to the example in Sect. 5, which uses sigmoid activated hidden layers only. Compare the weights and biases obtained by TensorFlow (or your ANN library of choice) with those obtained by your procedure after 200 epochs.

Appendix

Answers to Multiple Choice Questions

Question 1

Answer: 1, 2, 3, 4. All answers are found in the text.

Question 2

Answer: 1,2. A feedforward architecture is always convex w.r.t. each input variable if every activation function is convex and the weights are constrained to be either all positive or all negative. Simply using convex activation functions is not sufficient, since the composition of a convex function and the affine transformation of a convex function do not preserve the convexity. For example, if $\sigma(x) = x^2$, $w = -1$, and $b = 1$, then $\sigma(w\sigma(x) + b) = (-x^2 + 1)^2$ is not convex in x .

A feedforward architecture with positive weights is a monotonically increasing function of the input for any choice of monotonically increasing activation function.

The weights of a feedforward architecture need not be constrained for the output of a feedforward network to be bounded. For example, activating the output with a softmax function will bound the output. Only if the output is not activated, should the weights and bias in the final layer be bounded to ensure bounded output.

The bias terms in a network shift the output but also effect the derivatives of the output w.r.t. to the input when the layer is activated.

Question 3

Answer: 1,2,3,4. The training of a neural network involves minimizing a loss function w.r.t. the weights and biases over the training data. L_1 regularization is used during model selection to penalize models with too many parameters. The loss function is augmented with a Lagrange penalty for the number of weights. In deep learning, regularization can be applied to each layer of the network. Therefore each layer has an associated regularization parameter. Back-propagation uses the chain rule to update the weights of the network but is not guaranteed to converge to a unique minimum. This is because the loss function is not convex w.r.t. the weights. Stochastic gradient descent is a type of optimization method which is implemented with back-propagation. There are variants of SGD, however, such as adding Nestov's momentum term, ADAM , or RMSProp.

Back-Propagation

Let us consider a feedforward architecture with an input layer, $L - 1$ hidden layers, and one output layer, with K units in the output layer for classification of K categories. As a result, we have L sets of weights and biases $(W^{(\ell)}, \mathbf{b}^{(\ell)})$ for $\ell = 1, \dots, L$, corresponding to the layer inputs $Z^{(\ell-1)}$ and outputs $Z^{(\ell)}$ for $\ell = 1, \dots, L$. Recall that each layer is an activation of a semi-affine transformation, $I^{(\ell)}(Z^{(\ell-1)}) := W^{(\ell)}Z^{(\ell-1)} + b^{(\ell)}$. The corresponding activation functions are denoted as $\sigma^{(\ell)}$. The activation function for the output layer is a softmax function, $\sigma_s(x)$.

Here we use the cross-entropy as the loss function, which is defined as

$$\mathcal{L} := - \sum_{k=1}^K Y_k \log \hat{Y}_k.$$

The relationship between the layers, for $\ell \in \{1, \dots, L\}$ are

$$\hat{Y}(X) = Z^{(L)} = \sigma_s(I^{(L)}) \in [0, 1]^K,$$

$$Z^{(\ell)} = \sigma^{(\ell)}(I^{(\ell)}), \quad \ell = 1, \dots, L - 1,$$

$$Z^{(0)} = X.$$

The update rules for the weights and biases are

$$\Delta W^{(\ell)} = -\gamma \nabla_{W^{(\ell)}} \mathcal{L},$$

$$\Delta \mathbf{b}^{(\ell)} = -\gamma \nabla_{\mathbf{b}^{(\ell)}} \mathcal{L}.$$

We now begin the back-propagation, tracking the intermediate calculations carefully using Einstein summation notation.

For the gradient of \mathcal{L} w.r.t. $W^{(L)}$ we have

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{ij}^{(L)}} &= \sum_{k=1}^K \frac{\partial \mathcal{L}}{\partial Z_k^{(L)}} \frac{\partial Z_k^{(L)}}{\partial w_{ij}^{(L)}} \\ &= \sum_{k=1}^K \frac{\partial \mathcal{L}}{\partial Z_k^{(L)}} \sum_{m=1}^K \frac{\partial Z_k^{(L)}}{\partial I_m^{(L)}} \frac{\partial I_m^{(L)}}{\partial w_{ij}^{(L)}}\end{aligned}$$

But

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial Z_k^{(L)}} &= -\frac{Y_k}{Z_k^{(L)}} \\ \frac{\partial Z_k^{(L)}}{\partial I_m^{(L)}} &= \frac{\partial}{\partial I_m^{(L)}} [\sigma(I^{(L)})]_k \\ &= \frac{\partial}{\partial I_m^{(L)}} \frac{\exp[I_k^{(L)}]}{\sum_{n=1}^K \exp[I_n^{(L)}]} \\ &= \begin{cases} -\frac{\exp[I_k^{(L)}]}{\sum_{n=1}^K \exp[I_n^{(L)}]} \frac{\exp[I_m^{(L)}]}{\sum_{n=1}^K \exp[I_n^{(L)}]} & \text{if } k \neq m \\ \frac{\exp[I_k^{(L)}]}{\sum_{n=1}^K \exp[I_n^{(L)}]} - \frac{\exp[I_k^{(L)}]}{\sum_{n=1}^K \exp[I_n^{(L)}]} \frac{\exp[I_m^{(L)}]}{\sum_{n=1}^K \exp[I_n^{(L)}]} & \text{otherwise} \end{cases} \\ &= \begin{cases} -\sigma_k \sigma_m & \text{if } k \neq m \\ \sigma_k (1 - \sigma_m) & \text{otherwise} \end{cases} \\ &= \sigma_k (\delta_{km} - \sigma_m) \quad \text{where } \delta_{km} \text{ is the Kronecker's Delta} \\ \frac{\partial I_m^{(L)}}{\partial w_{ij}^{(L)}} &= \delta_{mi} Z_j^{(L-1)} \\ \implies \frac{\partial \mathcal{L}}{\partial w_{ij}^{(L)}} &= -\sum_{k=1}^K \frac{Y_k}{Z_k^{(L)}} \sum_{m=1}^K Z_m^{(L)} (\delta_{km} - Z_m^{(L)}) \delta_{mi} Z_j^{(L-1)} \\ &= -Z_j^{(L-1)} \sum_{k=1}^K Y_k (\delta_{ki} - Z_i^{(L)}) \\ &= Z_j^{(L-1)} (Z_i^{(L)} - Y_i),\end{aligned}$$

where we have used the fact that $\sum_{k=1}^K Y_k = 1$ in the last equality. Similarly for $\mathbf{b}^{(L)}$, we have

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b_i^{(L)}} &= \sum_{k=1}^K \frac{\partial \mathcal{L}}{\partial Z_k^{(L)}} \sum_{m=1}^K \frac{\partial Z_k^{(L)}}{\partial I_m^{(L)}} \frac{\partial I_m^{(L)}}{\partial b_i^{(L)}} \\ &= Z_i^{(L)} - Y_i\end{aligned}$$

It follows that

$$\begin{aligned}\nabla_{\mathbf{b}^{(L)}} \mathcal{L} &= Z^{(L)} - Y \\ \nabla_{W^{(L)}} \mathcal{L} &= \nabla_{\mathbf{b}^{(L)}} \mathcal{L} \otimes Z^{(L-1)},\end{aligned}$$

where \otimes denotes the outer product.

Now for the gradient of \mathcal{L} w.r.t. $W^{(L-1)}$ we have

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{ij}^{(L-1)}} &= \sum_{k=1}^K \frac{\partial L}{\partial Z_k^{(L)}} \frac{\partial Z_k^{(L)}}{\partial w_{ij}^{(L-1)}} \\ &= \sum_{k=1}^K \frac{\partial \mathcal{L}}{\partial Z_k^{(L)}} \sum_{m=1}^K \frac{\partial Z_k^{(L)}}{\partial I_m^{(L)}} \sum_{n=1}^{n^{(L-1)}} \frac{\partial I_m^{(L)}}{\partial Z_n^{(L-1)}} \sum_{p=1}^{n^{(L-1)}} \frac{\partial Z_n^{(L-1)}}{\partial I_p^{(L-1)}} \frac{\partial I_p^{(L-1)}}{\partial w_{ij}^{(L-1)}}\end{aligned}$$

If we assume that $\sigma^{(\ell)}(x) = \text{sigmoid}(x)$, $\ell \in \{1, \dots, L-1\}$, then

$$\begin{aligned}\frac{\partial I_m^{(L)}}{\partial Z_n^{(L-1)}} &= w_{mn}^{(L)} \\ \frac{\partial Z_n^{(L-1)}}{\partial I_p^{(L-1)}} &= \frac{\partial}{\partial I_p^{(L-1)}} \left(\frac{1}{1 + \exp(-I_n^{(L-1)})} \right) \\ &= \frac{1}{1 + \exp(-I_n^{(L-1)})} \frac{\exp(-I_n^{(L-1)})}{1 + \exp(-I_n^{(L-1)})} \delta_{np} \\ &= Z_n^{(L-1)} (1 - Z_n^{(L-1)}) \delta_{np} = \sigma_n^{(L-1)} (1 - \sigma_n^{(L-1)}) \delta_{np} \\ \frac{\partial I_p^{(L-1)}}{\partial w_{ij}^{(L-1)}} &= \delta_{pi} Z_j^{(L-2)} \\ \implies \frac{\partial L}{\partial w_{ij}^{(L)}} &= - \sum_{k=1}^K \frac{Y_k}{Z_k^{(L)}} \sum_{m=1}^K Z_k^{(L)} (\delta_{km} - Z_m^{(L)})\end{aligned}$$

$$\begin{aligned}
& \sum_{n=1}^{n^{(L-1)}} w_{mn}^{(L)} \sum_{p=1}^{n^{(L-1)}} Z_n^{(L-1)} (1 - Z_n^{(L-1)}) \delta_{np} \delta_{pi} Z_j^{(L-2)} \\
&= - \sum_{k=1}^K Y_k \sum_{m=1}^K (\delta_{km} - Z_m^{(L)}) \sum_{n=1}^{n^{(L-1)}} w_{mn}^{(L)} Z_n^{(L-1)} (1 - Z_n^{(L-1)}) \delta_{ni} Z_j^{(L-2)} \\
&= - \sum_{k=1}^K Y_k \sum_{m=1}^K (\delta_{km} - Z_m^{(L)}) w_{mi}^{(L)} Z_i^{(L-2)} (1 - Z_i^{(L-1)}) Z_j^{(L-2)} \\
&= - Z_j^{(L-2)} Z_i^{(L-1)} (1 - Z_i^{(L-1)}) \sum_{m=1}^K w_{mi}^{(L)} \sum_{k=1}^K (\delta_{km} Y_k - Z_m^{(L)} Y_k) \\
&= Z_j^{(L-2)} Z_i^{(L-1)} (1 - Z_i^{(L-1)}) (Z^{(L)} - Y)^T \mathbf{w}_{,i}^{(L)}
\end{aligned}$$

Similarly we have

$$\frac{\partial \mathcal{L}}{\partial b_i^{(L-1)}} = Z_i^{(L-1)} (1 - Z_i^{(L-1)}) (Z^{(L)} - Y)^T \mathbf{w}_{,i}^{(L)}.$$

It follows that we can define the following recursion relation for the loss gradient:

$$\begin{aligned}
\nabla_{b^{(L-1)}} \mathcal{L} &= Z^{(L-1)} \circ (\mathbf{1} - Z^{(L-1)}) \circ (W^{(L)}{}^T \nabla_{b^{(L)}} \mathcal{L}) \\
\nabla_{W^{(L-1)}} \mathcal{L} &= \nabla_{b^{(L-1)}} \mathcal{L} \otimes Z^{(L-2)} \\
&= Z^{(L-1)} \circ (\mathbf{1} - Z^{(L-1)}) \circ (W^{(L)}{}^T \nabla_{W^{(L)}} \mathcal{L}),
\end{aligned}$$

where \circ denotes the Hadamard product (element-wise multiplication). This recursion relation can be generalized for all layers and choice of activation functions. To see this, let the back-propagation error $\delta^{(\ell)} := \nabla_{b^{(\ell)}} \mathcal{L}$, and since

$$\begin{aligned}
\left[\frac{\partial \sigma^{(\ell)}}{\partial I^{(\ell)}} \right]_{ij} &= \frac{\partial \sigma_i^{(\ell)}}{\partial I_j^{(\ell)}} \\
&= \sigma_i^{(\ell)} (1 - \sigma_i^{(\ell)}) \delta_{ij}
\end{aligned}$$

or equivalently in matrix–vector form

$$\nabla_{I^{(\ell)}} \sigma^{(\ell)} = \text{diag}(\sigma^{(\ell)} \circ (\mathbf{1} - \sigma^{(\ell)})),$$

we can write, in general, for any choice of activation function for the hidden layer,

$$\delta^{(\ell)} = \nabla_{I^{(\ell)}} \sigma^{(\ell)} (W^{(\ell+1)})^T \delta^{(\ell+1)},$$

and

$$\nabla_{W^{(\ell)}} \mathcal{L} = \delta^{(\ell)} \otimes Z^{(\ell-1)}.$$

Proof of Theorem 4.2

Using the same deep structure shown in Fig. 4.9, Liang and Srikant (2016) find the binary expansion sequence $\{x_0, \dots, x_n\}$. In this step, they used n binary steps units in total. Then they rewrite $g_{m+1}(\sum_{i=0}^n \frac{x_i}{2^i})$,

$$\begin{aligned} g_{m+1}\left(\sum_{i=0}^n \frac{x_i}{2^i}\right) &= \sum_{j=0}^n \left[x_j \cdot \frac{1}{2^j} g_m\left(\sum_{i=0}^n \frac{x_i}{2^i}\right) \right] \\ &= \sum_{j=0}^n \max \left[2(x_j - 1) + \frac{1}{2^j} g_m\left(\sum_{i=0}^n \frac{x_i}{2^i}\right), 0 \right]. \end{aligned} \quad (4.57)$$

Clearly Eq. 4.57 defines iterations between the outputs of neighboring layers. Defining the output of the multilayer neural network as $\hat{f}(x) = \sum_{i=0}^p a_i g_i\left(\sum_{j=0}^n \frac{x_j}{2^j}\right)$. For this multilayer network, the approximation error is

$$\begin{aligned} |f(x) - \hat{f}(x)| &= \left| \sum_{i=0}^p a_i g_i\left(\sum_{j=0}^n \frac{x_j}{2^j}\right) - \sum_{i=0}^p a_i x^i \right| \\ &\leq \sum_{i=0}^p \left[|a_i| \cdot \left| g_i\left(\sum_{j=0}^n \frac{x_j}{2^j}\right) - x^i \right| \right] \leq \frac{p}{2^{n-1}}. \end{aligned}$$

This indicates, to achieve ε -approximation error, one should choose $n = \lceil \log \frac{p}{\varepsilon} \rceil + 1$. Besides, since $O(n + p)$ layers with $O(n)$ binary step units and $O(pn)$ ReLU units are used in total, this multilayer neural network thus has $O(p + \log \frac{p}{\varepsilon})$ layers, $O(\log \frac{p}{\varepsilon})$ binary step units, and $O(p \log \frac{p}{\varepsilon})$ ReLU units.

Table 4.2 Definitions of the functions $f(x)$ and $g(x)$

$f(x) := \max(x - \frac{1}{4}, 0),$ $\text{cI}_f = \{[0, \frac{1}{4}], (\frac{1}{4}, 1]\},$	$g(x) := \max(x - \frac{1}{2}, 0)$ $\text{cI}_g = \{[0, \frac{1}{2}], (\frac{1}{2}, 1]\}.$
--	---

Proof of Lemmas from Telgarsky (2016)

Proof (Proof of 4.1) Let cI_f denote the partition of \mathbb{R} corresponding to f , and cI_g denote the partition of \mathbb{R} corresponding to g .

First consider $f + g$, and moreover any intervals $U_f \in \text{cI}_f$ and $U_g \in \text{cI}_g$. Necessarily, $f + g$ has a single slope along $U_f \cap U_g$. Consequently, $f + g$ is $|\text{cI}|$ -sawtooth, where cI is the set of all intersections of intervals from cI_f and cI_g , meaning $\text{cI} := \{U_f \cap U_g : U_f \in \text{cI}_f, U_g \in \text{cI}_g\}$. By sorting the left endpoints of elements of cI_f and cI_g , it follows that $|\text{cI}| \leq k + l$ (the other intersections are empty).

For example, consider the example in Fig. 4.11 with partitions given in Table 4.2. The set of all intersections of intervals from cI_f and cI_g contains 3 elements:

$$\text{cI} = \{[0, \frac{1}{4}] \cap [0, \frac{1}{2}], (\frac{1}{4}, 1] \cap [0, \frac{1}{2}], (\frac{1}{4}, 1] \cap (\frac{1}{2}, 1]\} \quad (4.58)$$

Now consider $f \circ g$, and in particular consider the image $f(g(U_g))$ for some interval $U_g \in \text{cI}_g$. g is affine with a single slope along U_g ; therefore, f is being considered along a single unbroken interval $g(U_g)$. However, nothing prevents $g(U_g)$ from hitting all the elements of cI_f ; since U_g was arbitrary, it holds that $f \circ g$ is $(|\text{cI}_f| \cdot |\text{cI}_g|)$ -sawtooth. \square

Proof Recall the notation $\tilde{f}(x) := [f(x) \geq 1/2]$, whereby $\mathcal{E}(f) := \frac{1}{n} \sum_i [y_i \neq \tilde{f}(x_i)]$. Since f is piecewise monotonic with a corresponding partition \mathbb{R} having at most t pieces, then f has at most $2t - 1$ crossings of $1/2$: at most one within each interval of the partition, and at most 1 at the right endpoint of all but the last interval. Consequently, \tilde{f} is piecewise *constant*, where the corresponding partition of \mathbb{R} is into at most $2t$ intervals. This means n points with alternating labels must land in $2t$ buckets, thus the total number of points landing in buckets with at least three points is at least $n - 4t$. \square

Python Notebooks

The notebooks provided in the accompanying source code repository are designed to gain insight in toy classification datasets. They provide examples of deep feedforward classification, back-propagation, and Bayesian network classifiers. Further details of the notebooks are included in the README.md file.

References

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). Tensor flow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16 (pp. 265–283).
- Adams, R., Wallach, H., & Ghahramani, Z. (2010). Learning the structure of deep sparse graphical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (pp. 1–8).
- Andrews, D. (1989). A unified theory of estimation and inference for nonlinear dynamic models
a.r. gallant and h. white. *Econometric Theory*, 5(01), 166–171.
- Baillie, R. T., & Kapetanios, G. (2007). Testing for neglected nonlinearity in long-memory models. *Journal of Business & Economic Statistics*, 25(4), 447–461.
- Barber, D., & Bishop, C. M. (1998). Ensemble learning in Bayesian neural networks. *Neural Networks and Machine Learning*, 168, 215–238.
- Bartlett, P., Harvey, N., Liaw, C., & Mehrabian, A. (2017a). Nearly-tight VC-dimension bounds for piecewise linear neural networks. *CoRR*, *abs/1703.02930*.
- Bartlett, P., Harvey, N., Liaw, C., & Mehrabian, A. (2017b). Nearly-tight VC-dimension bounds for piecewise linear neural networks. *CoRR*, *abs/1703.02930*.
- Bengio, Y., Roux, N. L., Vincent, P., Delalleau, O., & Marcotte, P. (2006). Convex neural networks. In Y. Weiss, Schölkopf, B., & Platt, J. C. (Eds.), *Advances in neural information processing systems 18* (pp. 123–130). MIT Press.
- Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Berlin, Heidelberg: Springer-Verlag.
- Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015a, May). Weight uncertainty in neural networks. *arXiv:1505.05424 [cs, stat]*.
- Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015b). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.
- Chataigner, Crepe, & Dixon. (2020). *Deep local volatility*.
- Chen, J., Flood, M. D., & Sowers, R. B. (2017). Measuring the unmeasurable: an application of uncertainty quantification to treasury bond portfolios. *Quantitative Finance*, 17(10), 1491–1507.
- Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., et al. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems* (pp. 1223–1231).
- Dixon, M., Klabjan, D., & Bang, J. H. (2016). Classification-based financial markets prediction using deep neural networks. *CoRR*, *abs/1603.08604*.
- Feng, G., He, J., & Polson, N. G. (2018, Apr). Deep learning for predicting asset returns. *arXiv e-prints*, arXiv:1804.09314.
- Frey, B. J., & Hinton, G. E. (1999). Variational learning in nonlinear Gaussian belief networks. *Neural Computation*, 11(1), 193–213.
- Gal, Y. (2015). A theoretically grounded application of dropout in recurrent neural networks. *arXiv:1512.05287*.
- Gal, Y. (2016). *Uncertainty in deep learning*. Ph.D. thesis, University of Cambridge.
- Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international Conference on Machine Learning* (pp. 1050–1059).
- Gallant, A., & White, H. (1988, July). There exists a neural network that does not make avoidable mistakes. In *IEEE 1988 International Conference on Neural Networks* (vol.1 ,pp. 657–664).
- Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems* (pp. 2348–2356).
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction*. Springer.

- Heaton, J. B., Polson, N. G., & Witte, J. H. (2017). Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1), 3–12.
- Hernández-Lobato, J. M., & Adams, R. (2015). Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning* (pp. 1861–1869).
- Hinton, G. E., & Sejnowski, T. J. (1983). Optimal perceptual inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 448–453). IEEE New York.
- Hinton, G. E., & Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory* (pp. 5–13). ACM.
- Hornik, K., Stinchcombe, M., & White, H. (1989, July). Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5), 359–366.
- Horvath, B., Muguruza, A., & Tomas, M. (2019, Jan). *Deep learning volatility*. arXiv e-prints, arXiv:1901.09647.
- Hutchinson, J. M., Lo, A. W., & Poggio, T. (1994). A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 49(3), 851–889.
- Kingma, D. P., & Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.
- Kuan, C.-M., & White, H. (1994). Artificial neural networks: an econometric perspective. *Econometric Reviews*, 13(1), 1–91.
- Lawrence, N. (2005). Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6(Nov), 1783–1816.
- Liang, S., & Srikant, R. (2016). Why deep neural networks? *CoRR abs/1610.04161*.
- Lo, A. (1994). Neural networks and other nonparametric techniques in economics and finance. In *AIMR Conference Proceedings*, Number 9.
- MacKay, D. J. (1992a). A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3), 448–472.
- MacKay, D. J. C. (1992b, May). A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3), 448–472.
- Martin, C. H., & Mahoney, M. W. (2018). Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *CoRR abs/1810.01075*.
- Mhaskar, H., Liao, Q., & Poggio, T. A. (2016). Learning real and Boolean functions: When is deep better than shallow. *CoRR abs/1603.00988*.
- Mnih, A., & Gregor, K. (2014). Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*.
- Montúfar, G., Pascanu, R., Cho, K., & Bengio, Y. (2014, Feb). On the number of linear regions of deep neural networks. *arXiv e-prints*, arXiv:1402.1869.
- Mullainathan, S., & Spiess, J. (2017). Machine learning: An applied econometric approach. *Journal of Economic Perspectives*, 31(2), 87–106.
- Neal, R. M. (1990). *Learning stochastic feedforward networks*, Vol. 64. Technical report, Department of Computer Science, University of Toronto.
- Neal, R. M. (1992). *Bayesian training of backpropagation networks by the hybrid Monte Carlo method*. Technical report, CRG-TR-92-1, Dept. of Computer Science, University of Toronto.
- Neal, R. M. (2012). *Bayesian learning for neural networks*, Vol. 118. Springer Science & Business Media. bibtex: aneal2012bayesian.
- Nesterov, Y. (2013). *Introductory lectures on convex optimization: A basic course*, Volume 87. Springer Science & Business Media.
- Poggio, T. (2016). Deep learning: mathematics and neuroscience. A sponsored supplement to science brain-inspired intelligent robotics: The intersection of robotics and neuroscience, pp. 9–12.
- Polson, N., & Rockova, V. (2018, Mar). Posterior concentration for sparse deep learning. *arXiv e-prints*, arXiv:1803.09138.
- Polson, N. G., Willard, B. T., & Heidari, M. (2015). A statistical theory of deep learning via proximal splitting. *arXiv:1509.06061*.

- Racine, J. (2001). On the nonlinear predictability of stock returns using financial and economic variables. *Journal of Business & Economic Statistics*, 19(3), 380–382.
- Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.
- Ruiz, F. R., Aueb, M. T. R., & Blei, D. (2016). The generalized reparameterization gradient. In *Advances in Neural Information Processing Systems* (pp. 460–468).
- Salakhutdinov, R. (2008). *Learning and evaluating Boltzmann machines*. Tech. Rep., Technical Report UTML TR 2008-002, Department of Computer Science, University of Toronto.
- Salakhutdinov, R., & Hinton, G. (2009). Deep Boltzmann machines. In *Artificial Intelligence and Statistics* (pp. 448–455).
- Saul, L. K., Jaakkola, T., & Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4, 61–76.
- Sirignano, J., Sadhwani, A., & Giesecke, K. (2016, July). Deep learning for mortgage risk. *ArXiv e-prints*.
- Smolensky, P. (1986). *Parallel distributed processing: explorations in the microstructure of cognition* (Vol. 1. pp. 194–281). Cambridge, MA, USA: MIT Press.
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), 1929–1958.
- Swanson, N. R., & White, H. (1995). A model-selection approach to assessing the information in the term structure using linear models and artificial neural networks. *Journal of Business & Economic Statistics*, 13(3), 265–275.
- Telgarsky, M. (2016). Benefits of depth in neural networks. *CoRR abs/1602.04485*.
- Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning* (pp. 1064–1071). ACM.
- Tishby, N., & Zaslavsky, N. (2015). Deep learning and the information bottleneck principle. *CoRR abs/1503.02406*.
- Tran, D., Hoffman, M. D., Saurous, R. A., Brevdo, E., Murphy, K., & Blei, D. M. (2017, January). Deep probabilistic programming. *arXiv:1701.03757 [cs, stat]*.
- Vapnik, V. N. (1998). *Statistical learning theory*. Wiley-Interscience.
- Welling, M., Rosen-Zvi, M., & Hinton, G. E. (2005). Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems* (pp. 1481–1488).
- Williams, C. K. (1997). Computing with infinite networks. In *Advances in Neural Information Processing systems* (pp. 295–301).

Chapter 5

Interpretability



This chapter presents a method for interpreting neural networks which imposes minimal restrictions on the neural network design. The chapter demonstrates techniques for interpreting a feedforward network, including how to rank the importance of the features. An example demonstrating how to apply interpretability analysis to deep learning models for factor modeling is also presented.

1 Introduction

Once the neural network has been trained, a number of important issues surface around how to interpret the model parameters. This aspect is a prominent issue for practitioners in deciding whether to use neural networks in favor of other machine learning and statistical methods for estimating factor realizations, sometimes even if the latter's predictive accuracy is inferior.

In this section, we shall introduce a method for interpreting multilayer perceptrons which imposes minimal restrictions on the neural network design.

Chapter Objectives

By the end of this chapter, the reader should expect to accomplish the following:

- Apply techniques for interpreting a feedforward network, including how to rank the importance of the features.
- Learn how to apply interpretability analysis to deep learning models for factor modeling.

2 Background on Interpretability

There are numerous techniques for interpreting machine learning methods which treat the model as a black-box. A good example are Partial Dependence Plots (PDPs) as described by Greenwell et al. (2018). Other approaches also exist in the literature. Garson (1991) partitions hidden-output connection weights into components associated with each input neuron using absolute values of connection weights. Olden and Jackson (2002) determine the relative importance, $[R]_{ij}$, of the i th output to the j th predictor variable of the model as a function of the weights, according to a simple linear expression.

We seek to understand the limitations on the choice of activation functions and understand the effect of increasing layers and numbers of neurons on probabilistic interpretability. For example, under standard Gaussian i.i.d. data, how robust are the model's estimate of the importance of each input variable with variable number of neurons?

2.1 Sensitivities

We shall therefore turn to a “white-box” technique for determining the importance of the input variables. This approach generalizes Dimopoulos et al. (1995) to a deep neural network with interaction terms. Moreover, the method is directly consistent with how coefficients are interpreted in linear regression—they are *model* sensitivities. Model sensitivities are the change of the fitted model output w.r.t. input.

As a control, we shall use this property to empirically evaluate how reliably neural networks, even deep networks, learn data from a linear model.

Such an approach is appealing to practitioners who are evaluating the comparative performance of linear regression with neural networks and need the assurance that a neural network model is at least able to reproduce and match the coefficients on a linear dataset.

We also offset the common misconception that the activation functions must be deactivated for a neural network model to produce a linear output. Under linear data, any non-linear statistical model should be able to reproduce a statistical linear model under some choice of parameter values. Irrespective of whether data is linear or non-linear in practice - the best control experiment for comparing a neural network estimator with an OLS estimator is to simulate data under a linear regression model. In this scenario, the correct model coefficients are known and the error in the coefficient estimator can be studied.

To evaluate fitted model sensitivities analytically, we require that the function $\hat{Y} = f(X)$ is continuous and differentiable everywhere. Furthermore, for stability of

the interpretation, we shall require that $f(x)$ is Lipschitz continuous.¹ That is, there is a positive real constant K s.t. $\forall x_1, x_2 \in \mathbb{R}^p$, $|F(x_1) - F(x_2)| \leq K|x_1 - x_2|$. Such a constraint is necessary for the first derivative to be bounded and hence amenable to the derivatives, w.r.t. to the inputs, providing interpretability.

Fortunately, provided that the weights and biases are finite, each semi-affine function is Lipschitz continuous everywhere. For example, the function $\tanh(x)$ is continuously differentiable and its derivative $1 - \tanh^2(x)$ is globally bounded. With finite weights, the composition of $\tanh(x)$ with an affine function is also Lipschitz. Clearly $\text{ReLU}(x) := \max(\cdot, 0)$ is not continuously differentiable and one cannot use the approach described here. Note that for the following examples, we are indifferent to the choice of homoscedastic or heteroscedastic error, since the model sensitivities are independent of the error.

3 Explanatory Power of Neural Networks

In a linear regression model

$$\hat{Y} = F_{\beta}(X) := \beta_0 + \beta_1 X_1 + \cdots + \beta_K X_K, \quad (5.1)$$

the model sensitivities are

$$\partial_{X_i} \hat{Y} = \beta_i. \quad (5.2)$$

In a feedforward neural network, we can use the chain rule to obtain the model sensitivities

$$\partial_{X_i} \hat{Y} = \partial_{X_i} F_{W,b}(X) = \partial_{X_i} \sigma_{W^{(L)}, b^{(L)}}^{(L)} \circ \cdots \circ \sigma_{W^{(1)}, b^{(1)}}^{(1)}(X). \quad (5.3)$$

For example, with one hidden layer, $\sigma(x) := \tanh(x)$ and $\sigma_{W^{(1)}, b^{(1)}}^{(1)}(X) := \sigma(I^{(1)}) := \sigma(W^{(1)}X + b^{(1)})$:

$$\partial_{X_j} \hat{Y} = \sum_i \mathbf{w}_{,i}^{(2)} (1 - \sigma^2(I_i^{(1)})) w_{ij}^{(1)} \quad \text{where } \partial_x \sigma(x) = (1 - \sigma^2(x)). \quad (5.4)$$

In matrix form, with general σ , the Jacobian² of σ w.r.t X is $J = D(I^{(1)})W^{(1)}$ of σ ,

$$\partial_X \hat{Y} = W^{(2)} J(I^{(1)}) = W^{(2)} D(J^{(1)}) W^{(1)}, \quad (5.5)$$

¹If Lipschitz continuity is not imposed, then a small change in one of the input values could result in an undesirable large variation in the derivative.

²When σ is an identity function, the Jacobian $J(I^{(1)}) = W^{(1)}$.

where $D_{ii}(I) = \sigma'(I_i)$, $D_{ij} = 0$, $i \neq j$ is a diagonal matrix. Bounds on the sensitivities are given by the product of the weight matrices

$$\min(W^{(2)}W^{(1)}, 0) \leq \partial_X \hat{Y} \leq \max(W^{(2)}W^{(1)}, 0). \quad (5.6)$$

3.1 Multiple Hidden Layers

The model sensitivities can be readily generalized to an L layer deep network by evaluating the Jacobian matrix:

$$\partial_X \hat{Y} = W^{(L)} J(I^{(L-1)}) = W^{(L)} D(I^{(L-1)}) W^{(L-1)} \dots D(I^{(1)}) W^{(1)}. \quad (5.7)$$

3.2 Example: Step Test

To illustrate our interpretability approach, we shall consider a simple example. The model is trained to the following data generation process where the coefficients of the features are stepped and the error, here, is i.i.d. uniform:

$$\hat{Y} = \sum_{i=1}^{10} i X_i, \quad X_i \sim \mathcal{U}(0, 1). \quad (5.8)$$

Figure 5.1 shows the ranked importance of the input variables in a neural network with one hidden layer. Our interpretability method is compared with well-known black-box interpretability methods such as Garson's algorithm (Garson 1991) and Olden's algorithm (Olden and Jackson 2002). Our approach is the only technique to interpret the fitted neural network which is consistent with how a linear regression model would interpret the input variables.

4 Interaction Effects

The previous example is too simplistic to illustrate another important property of our interpretability method, namely the ability to capture pairwise interaction terms. The pairwise interaction effects are readily available by evaluating the elements of the Hessian matrix. For example, with one hidden layer, the Hessian takes the form:

$$\partial_{X_i X_j}^2 \hat{Y} = W^{(2)} \text{diag}(W_i^{(1)}) D'(I^{(1)}) W_j^{(1)}, \quad (5.9)$$

where it is assumed that the activation function is at least twice differentiable everywhere, e.g. $\tanh(x)$.

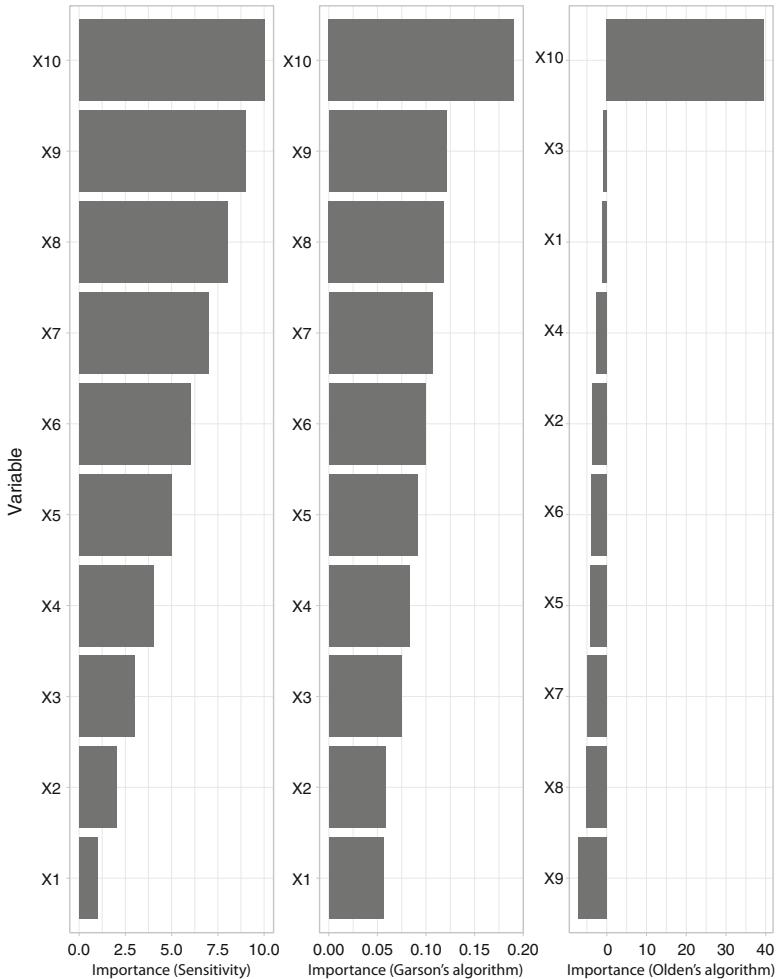


Fig. 5.1 Step test: This figure shows the ranked importance of the input variables in a neural network with one hidden layer. (left) Our sensitivity based approach for input interpretability. (Center) Garson's algorithm and (Right) Olden's algorithm. Our approach is the only technique to interpret the fitted neural network which is consistent with how a linear regression model would interpret the input variables

4.1 Example: Friedman Data

To illustrate our input variable and interaction effect ranking approach, we will use a classical nonlinear benchmark regression problem. The input space consists of ten i.i.d. uniform $\mathcal{U}(0, 1)$ random variables; however, only five out of these ten actually appear in the true model. The response is related to the inputs according to the formula

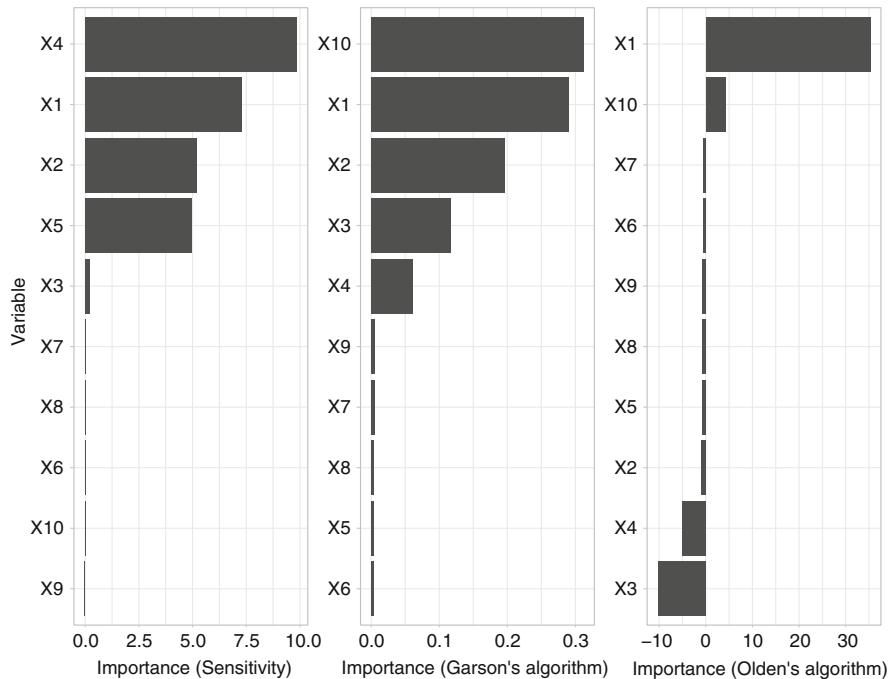


Fig. 5.2 Friedman test: Ranked model sensitivities of the fitted neural network to the input. (left) Our sensitivity based approach for input interpretability. (Center) Garson's algorithm and (Right) Olden's algorithm

$$Y = 10 \sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5 + \epsilon,$$

using white noise error, $\epsilon \sim \mathcal{N}(0, \sigma^2)$. We fit a NN with one hidden layer containing eight units and a weight decay of 0.01 (these parameters were chosen using 5-fold cross-validation) to 500 observations simulated from the above model with $\sigma = 1$. The cross-validated R^2 value was 0.94.

Figures 5.2 and 5.3, respectively, compare the ranked model sensitivities and ranked interaction terms of the fitted neural network with Garson's and Olden's algorithm.

5 Bounds on the Variance of the Jacobian

General results on the bound of the variance of the Jacobian for any activation function are difficult to derive. However, we derive the following result for a *ReLU* activated single-layer feedforward network. In matrix form, with $\sigma(x) = \max(x, 0)$, the Jacobian, J , can be written as a linear combination of Heaviside functions:

$$J := J(X) = \partial_X \hat{Y}(X) = W^{(2)} J(I^{(1)}) = W^{(2)} H(W^{(1)} X + b^{(1)}) W^{(1)}, \quad (5.10)$$

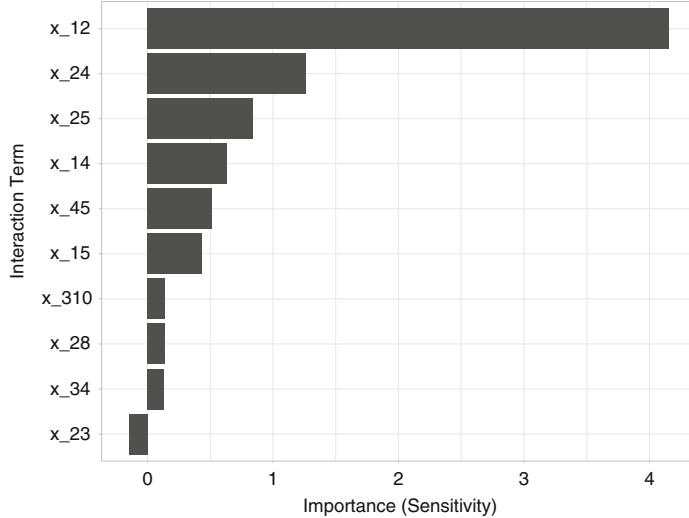


Fig. 5.3 Friedman test: Ranked pairwise interaction terms in the fitted neural network to the input. (Left) Our sensitivity based approach for ranking interaction terms. (Center) Garson’s algorithm and (Right) Olden’s algorithm

where $H_{ii}(Z) = H(I_i^{(1)}) = \mathbb{1}_{\{I_i^{(1)} > 0\}}$, $H_{ij} = 0$, $j \geq i$. We assume that the mean of the Jacobian is independent of the number of hidden units, $\mu_{ij} := \mathbb{E}[J_{ij}]$. Then we can state the following bound on the Jacobian of the network for the special case when the input is one-dimensional.

Theorem (Dixon and Polson 2019) *If $X \in \mathbb{R}^p$ is i.i.d. and there are n hidden units, with ReLU activation i , then the variance of a single-layer feedforward network with K outputs is bounded by μ_{ij}*

$$\mathbb{V}[J_{ij}] = \mu_{ij} \frac{n-1}{n} < \mu_{ij}, \quad \forall i \in \{1, \dots, K\} \text{ and } \forall j \in \{1, \dots, p\}. \quad (5.11)$$

See Appendix “Proof of Variance Bound on Jacobian” for the proof.

Remark 5.1 The theorem establishes a negative result for a ReLU activated shallow network—increasing the number of hidden units, increases the bound on the variance of the Jacobian, and hence reduces interpretability of the sensitivities. Note that if we do not assume that the mean of the Jacobian is fixed under varying n , then we have the more general bound:

$$\mathbb{V}[J_{ij}] \leq \mu_{ij}, \quad (5.12)$$

and hence the effect of network architecture on the bound of the variance of the Jacobian is not clear. Note that the theorem holds without (i) distributional assumptions on X other than i.i.d. data and (ii) specifying the number of data points. \square

Remark 5.2 This result also suggests that the inputs should be rescaled so that each μ_{ij} , the expected value of the Jacobian, is a small positive value, although it may not be possible to find such a scaling for all (i, j) pairs. \square

5.1 Chernoff Bounds

We can derive probabilistic bounds on the Jacobians for any choice of activation function. Let $\delta > 0$ and a_1, \dots, a_{n-1} be reals in $(0, 1]$. Let X_1, \dots, X_{n-1} be independent Bernoulli trials with $\mathbb{E}[X_k] = p_k$ so that

$$\mathbb{E}[J] = \sum_{k=1}^{n-1} a_k p_k = \mu. \quad (5.13)$$

The Chernoff-type bound exists on deviations of J above the mean

$$\Pr(J > (1 + \delta)\mu) = \left[\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right]^\mu. \quad (5.14)$$

A similar bound exists for deviations of J below the mean. For $\gamma \in (0, 1]$:

$$\Pr(J - \mu < -\gamma\mu) < \left[\frac{e^\gamma}{(1 + \gamma)^{1+\gamma}} \right]^\mu. \quad (5.15)$$

These bounds are generally weak and are suited to large deviations, i.e. the tail regions. The bounds are shown in the Fig. 5.4 for different values of μ . Here, μ is increasing towards the upper right-hand corner of the plot.

5.2 Simulated Example

In this section, we demonstrate the estimation properties of neural network sensitivities applied to data simulated from a linear model. We show that the sensitivities in a neural network are consistent with the linear model, even if the neural network model is non-linear. We also show that the confidence intervals, estimated by sampling, converge with increasing hidden units.

We generate 400 simulated training samples from the following linear model with i.i.d. Gaussian error:

Fig. 5.4 The Chernoff-type bounds for deviations of J above the mean, μ . Various μ are shown in the plot, with μ increasing towards the upper right-hand corner of the plot

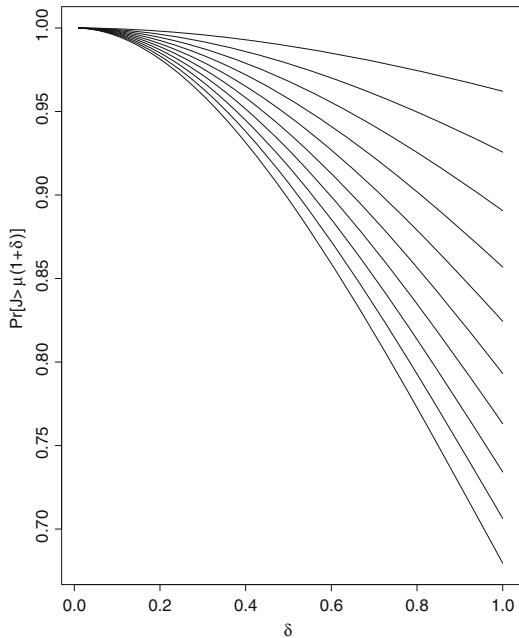


Table 5.1 This table compares the functional form of the variable sensitivities and values with an OLS estimator. NN_0 is a zero hidden layer feedforward network and NN_1 is a one hidden layer feedforward network with 10 hidden neurons and tanh activation functions

Model	Intercept		Sensitivity of X_1		Sensitivity of X_2	
OLS	$\hat{\beta}_0$	0.011	$\hat{\beta}_1$	1.015	$\hat{\beta}_2$	1.018
NN_0	$\hat{b}^{(1)}$	0.020	$\hat{W}_1^{(1)}$	1.018	$\hat{W}_2^{(1)}$	1.021
NN_1	$\hat{W}^{(2)}\sigma(\hat{b}^{(1)}) + \hat{b}^{(2)}$	0.021	$\mathbb{E}[\hat{W}^{(2)}D(I^{(1)})\hat{W}_1^{(1)}]$	1.014	$\mathbb{E}[\hat{W}^{(2)}D(I^{(1)})\hat{W}_2^{(1)}]$	1.022

$$Y = \beta_1 X_1 + \beta_2 X_2 + \epsilon, \quad X_1, X_2, \epsilon \sim N(0, 1), \quad \beta_1 = 1, \beta_2 = 1. \quad (5.16)$$

Table 5.1 compares an OLS estimator with a zero hidden layer feedforward network (NN_0) and a one hidden layer feedforward network with 10 hidden neurons and tanh activation functions (NN_1). The functional form of the first two regression models is equivalent, although the OLS estimator has been computed using a matrix solver, whereas the zero layer hidden network parameters have been fitted with stochastic gradient descent.

The fitted parameters values will vary slightly with each optimization as the stochastic gradient descent is randomized. However, the sensitivity terms are given in closed form and easily mapped to the linear model. In an industrial setting, such a one-to-one mapping is useful for migrating to a deep factor model where, for model validation purposes, compatibility with linear models should be recovered in a limiting case. Clearly, if the data is not generated from a linear model, then the parameter values would vary across models.

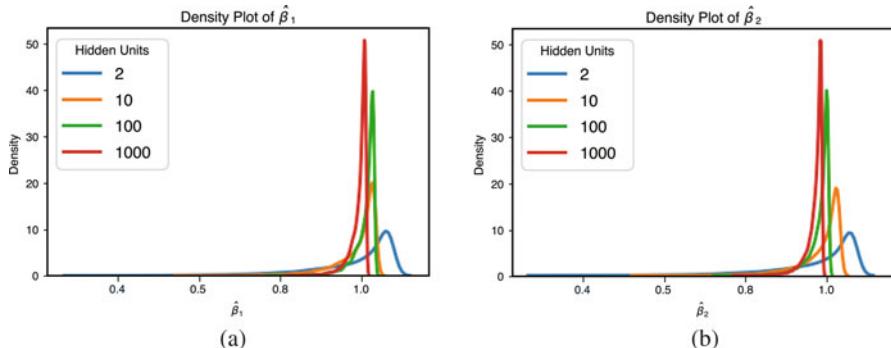


Fig. 5.5 This figure shows the empirical distribution of the sensitivities $\hat{\beta}_1$ and $\hat{\beta}_2$. The sharpness of the distribution is observed to converge with the number of hidden units. (a) Density of $\hat{\beta}_1$. (b) Density of $\hat{\beta}_2$

Table 5.2 This table shows the moments and 99% confidence interval of the empirical distribution of the sensitivity $\hat{\beta}_1$. The sharpness of the distribution is observed to converge monotonically with the number of hidden units

Hidden Units	Mean	Median	Std.dev	1% C.I.	99% C.I.
2	0.980875	1.0232913	0.10898393	0.58121675	1.0729908
10	0.9866159	1.0083131	0.056483902	0.76814914	1.0322522
50	0.99183553	1.0029879	0.03123002	0.8698967	1.0182846
100	1.0071343	1.0175397	0.028034585	0.89689034	1.0296803
200	1.0152218	1.0249312	0.026156902	0.9119074	1.0363332

Table 5.3 This table shows the moments and the 99% confidence interval of the empirical distribution of the sensitivity $\hat{\beta}_2$. The sharpness of the distribution is observed to converge monotonically with the number of hidden units

Hidden Units	Mean	Median	Std.dev	1% C.I.	99% C.I.
2	0.98129386	1.0233982	0.10931312	0.5787732	1.073728
10	0.9876832	1.0091512	0.057096474	0.76264584	1.0339714
50	0.9903236	1.0020974	0.031827927	0.86471796	1.0152498
100	0.9842479	0.9946766	0.028286876	0.87199813	1.0065105
200	0.9976638	1.0074166	0.026751818	0.8920307	1.0189484

Figure 5.5 and Tables 5.2 and 5.3 show the empirical distribution of the fitted sensitivities using the single hidden layer model with increasing hidden units. The sharpness of the distributions is observed to converge monotonically with the number of hidden units. The confidence intervals are estimated under a non-parametric distribution.

In general, provided the weights and biases of the network are finite, the variances of the sensitivities are bounded for any input and choice of activation function.

We do not recommend using ReLU activation because it does not permit identification of the interaction terms and has provably non-convergent sensitivity variances as a function of the number of hidden units (see Appendix “Proof of Variance Bound on Jacobian”).

6 Factor Modeling

Rosenberg and Marathe (1976) introduced a cross-sectional fundamental factor model to capture the effects of macroeconomic events on individual securities. The choice of factors are microeconomic characteristics—essentially common factors, such as industry membership, financial structure, or growth orientation (Nielsen and Bender 2010).

The BARRA fundamental factor model expresses the linear relationship between K fundamental factors and N asset returns:

$$\mathbf{r}_t = B_t \mathbf{f}_t + \boldsymbol{\epsilon}_t, \quad t = 1, \dots, T, \quad (5.17)$$

where $B_t = [\mathbf{1} \mid \boldsymbol{\beta}_1(t) \mid \dots \mid \boldsymbol{\beta}_K(t)]$ is the $N \times K+1$ matrix of known factor loadings (betas): $\beta_{i,k}(t) := (\boldsymbol{\beta}_k)_i(t)$ is the exposure of asset i to factor k at time t .

The factors are asset specific attributes such as market capitalization, industry classification, style classification. $\mathbf{f}_t = [\alpha_t, f_{1,t}, \dots, f_{K,t}]$ is the $K+1$ vector of unobserved factor realizations at time t , including α_t .

\mathbf{r}_t is the N -vector of asset returns at time t . The errors are assumed independent of the factor realizations $\rho(f_{i,t}, \epsilon_{j,t}) = 0, \forall i, j, t$ with Gaussian error, $\mathbb{E}[\epsilon_{j,t}^2] = \sigma^2$.

6.1 Non-linear Factor Models

We can extend the linear model to a non-linear cross-sectional fundamental factor model of the form

$$\mathbf{r}_t = F_t(B_t) + \boldsymbol{\epsilon}_t, \quad (5.18)$$

where \mathbf{r}_t are asset returns, $F_t : \mathbb{R}^K \rightarrow \mathbb{R}$ is a differentiable non-linear function that maps the i th row of B to the i th asset return at time t . The map is assumed to incorporate a bias term so that $F_t(\mathbf{0}) = \alpha_t$. In the special case when $F_t(B_t)$ is linear, the map is $F_t(B_t) = B_t \mathbf{f}_t$.

A key feature is that we do not assume that $\boldsymbol{\epsilon}_t$ is described by a parametric distribution, such as a Gaussian distribution. In our example, we shall treat $\boldsymbol{\epsilon}_t$ as i.i.d., however, we can extend the methodology to non-i.i.d. idea as in Dixon and Polson (2019). In our setup, the model shall just be used to predict the next period returns only and stationarity of the factor realizations is not required.

We approximate a non-linear map, $F_t(B_t)$, with a feedforward neural network cross-sectional factor model:

$$\mathbf{r}_t = F_{W_t, b_t}(B_t) + \boldsymbol{\epsilon}_t, \quad (5.19)$$

where F_{W_t, b_t} is a deep neural network with L layers.

6.2 Fundamental Factor Modeling

This section presents an application of deep learning to a toy fundamental factor model. Factor models in practice include significantly more fundamental factors than used here. But the purpose, here, is to illustrate the application of interpretable deep learning to financial data.

We define the universe as the top 250 stocks from the S&P 500 index, ranked by market cap. Factors are given by Bloomberg and reported monthly over a hundred month period beginning in February 2008. We remove stocks with too many missing factor values, leaving 218 stocks.

The historical factors are inputs to the model and are standardized to enable model interpretability. These factors are (i) current enterprise value; (ii) Price-to-Book ratio; (iii) current enterprise value to trailing 12 month EBITDA; (iv) Price-to-Sales ratio; (v) Price-to-Earnings ratio; and (vi) log market cap. The responses are the monthly asset returns for each stock in our universe based on the daily adjusted closing prices of the stocks.

We use Tensorflow (Abadi et al. 2016) to implement a two hidden layer feedforward network and develop a custom implementation for the least squares error and variable sensitivities and is available in the deep factor models notebook. The OLS regression is implemented by the Python `StatsModels` module.

All deep learning results are shown using L_1 regularization and \tanh activation functions. The number of hidden units and regularization parameters are found by three-fold cross-validation and reported alongside the results.

Figure 5.6 compares the performance of an OLS estimator with the feedforward neural network with 10 hidden units in the first hidden layer and 10 hidden units in the second layer and $\lambda_1 = 0.001$.

Figure 5.7 shows the in-sample MSE as a function of the number of hidden units in the hidden layer. The neural networks are trained here without L_1 regularization to demonstrate the effect of solely increasing the number of hidden units in the first layer. Increasing the number of hidden units reduces the bias in the model.

Figure 5.8 shows the effect of L_1 regularization on the MSE errors for a network with 10 units in each of the two hidden layers. Increasing the level of L_1 regularization increases the in-sample bias but reduces the out-of-sample bias, and hence the variance of the estimation error.

Figure 5.9 compares the distribution of sensitivities to each factor over the entire 100 month period using the neural network (top) and OLS regression (bottom). The

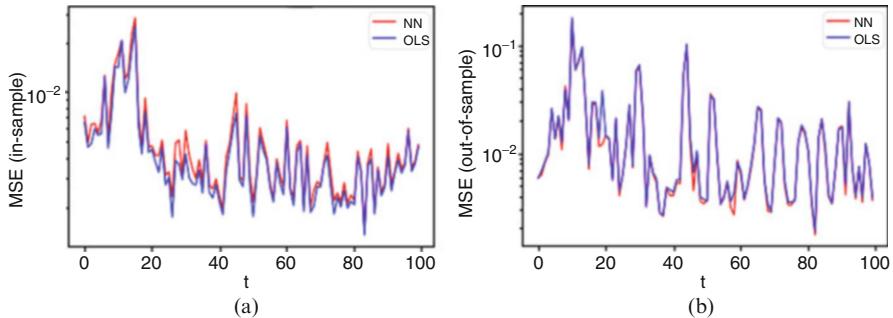


Fig. 5.6 This figure compares the in-sample and out-of-sample performances of an OLS estimator (OLS) with a feedforward neural network (NN), as measured by the mean squared error (MSE). The neural network is observed to always exhibit slightly lower out-of-sample MSE, although the effect of deep networks on this problem is marginal because the dataset is too simplistic. **(a)** In-sample error. **(b)** Out-of-sample error

Fig. 5.7 This figure shows the in-sample MSE as a function of the number of hidden units in the hidden layer. Increasing the number of hidden units reduces the bias in the model

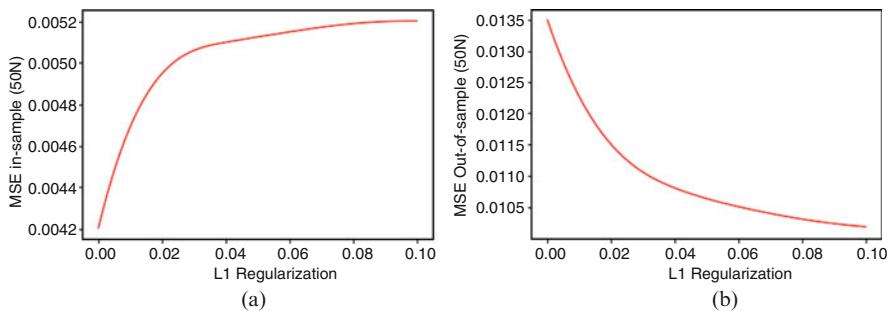
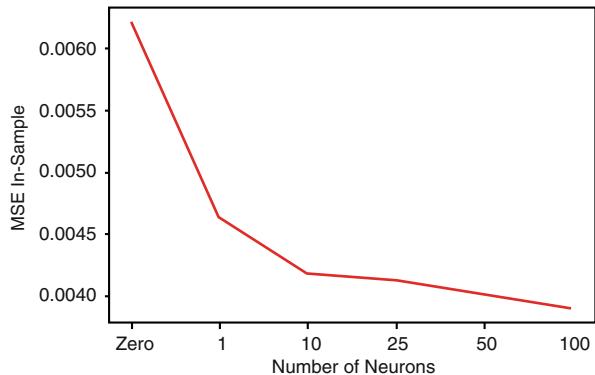


Fig. 5.8 These figures show the effect of L_1 regularization on the MSE errors for a network with 10 neurons in each of the two hidden layers. **(a)** In-sample. **(b)** Out-of-sample

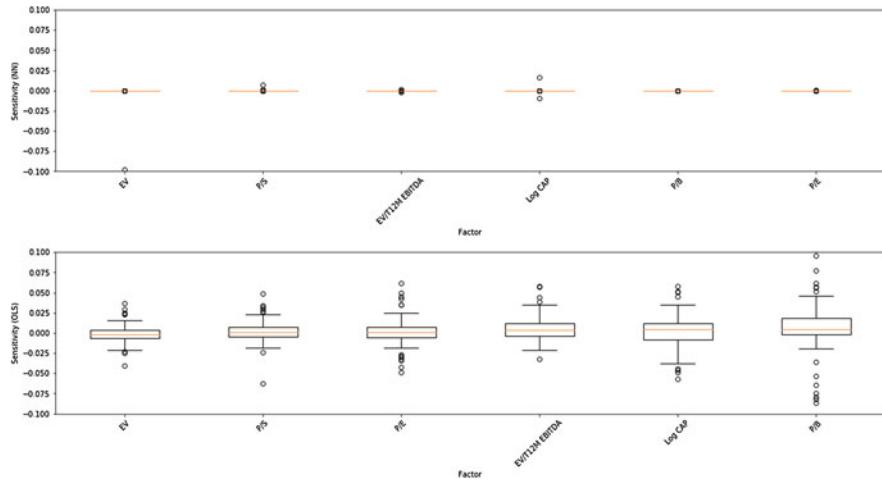


Fig. 5.9 The distribution of sensitivities to each factor over the entire 100 month period using the neural network (top). The sensitivities are sorted in ascending order from left to right by their median values. The same sensitivities using OLS linear regression (bottom)

sensitivities are sorted in ascending order from left to right by their median values. We observe that the OLS regression is much more sensitive to the factors than the NN. We further note that the NN ranks the top sensitivities differently to OLS.

Clearly, the above results are purely illustrative of the interpretability methodology and not intended to be representative of a real-world factor model. Such a choice of factors is observed to provide little benefit on the information ratios of a simple stock selection strategy.

Larger Dataset

For completeness, we provide evidence that our neural network factor model generates positive and higher information ratios than OLS when used to sort portfolios from a larger universe, using up to 50 factors (see Table 5.4 for a description of the factors). The dataset is not provided due to data licensing restrictions.

We define the universe as 3290 stocks from the Russell 3000 index. Factors are given by Bloomberg and reported monthly over the period from November 2008 to November 2018. We train a two-hidden layer deep network with 50 hidden units using ReLU activation.

Figure 5.10 compares the out-of-sample performance of neural networks and OLS regression by the MSE (left) and the information ratios of a portfolio selection strategy which selects the n stocks with the highest predicted monthly returns (right). The information ratios are evaluated for various size portfolios, using the Russell 3000 index as the benchmark. Also shown, for control, are randomly selected portfolios.

Table 5.4 A short description of the factors used in the Russell 3000 deep learning factor model demonstrated at the end of this chapter

<i>Value factors</i>	
B/P	Book to price
CF/P	Cash flow to price
E/P	Earning to price
S/EV	Sales to enterprise value (EV). EV is given by EV=Market Cap + LT Debt + max(ST Debt-Cash,0), where LT (ST) stands for long (short) term
EB/EV	EBIDTA to EV
FE/P	Forecasted E/P. Forecast earnings are calculated from Bloomberg earnings consensus estimates data For coverage reasons, Bloomberg uses the 1-year and 2-year forward earnings
DIV	Dividend yield. The exposure to this factor is just the most recently announced annual net dividends divided by the market price Stocks with high dividend yields have high exposures to this factor
<i>Size factors</i>	
MC	Log (Market capitalization)
S	Log (sales)
TA	Log (total assets)
<i>Trading activity factors</i>	
TrA	Trading activity is a turnover based measure Bloomberg focuses on turnover which is trading volume normalized by shares outstanding This indirectly controls for the Size effect The exponential weighted average (EWMA) of the ratio of shares traded to shares outstanding In addition, to mitigate the impacts of those sharp short-lived spikes in trading volume, Bloomberg winsorizes the data first daily trading volume data is compared to the long-term EWMA volume(180 day half-life), then the data is capped at 3 standard deviations away from the EWMA average
<i>Earnings variability factors</i>	
EaV/TA	Earnings volatility to total assets Earnings volatility is measured over the last 5 years/median total assets over the last 5 years
CFV/TA	Cash flow volatility to total assets Cash flow volatility is measured over the last 5 years/median total assets over the last 5 years
SV/TA	Sales volatility to total assets Sales volatility over the last 5 years/median total assets over the last 5 year

(continued)

Table 5.4 (continued)

	<i>Volatility factors</i>
RV	Rolling volatility which is the return volatility over the latest 252 trading days
CB	Rolling CAPM beta which is the regression coefficient from the rolling window regression of stock returns on local index returns
	<i>Growth factors</i>
TAG	Total asset growth is the 5-year average growth in total assets divided by the average total assets over the last 5 years
EG	Earnings growth is the 5-year average growth in earnings divided by the average total assets over the last 5 years
	<i>GSIC sectorial codes</i>
(I)ndustry	{10, 20, 30, 40, 50, 60, 70}
(S)ub-(I)ndustry	{10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80}
(S)ector	{10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60}
(I)ndustry (G)roup	{10, 20, 30, 40, 50}

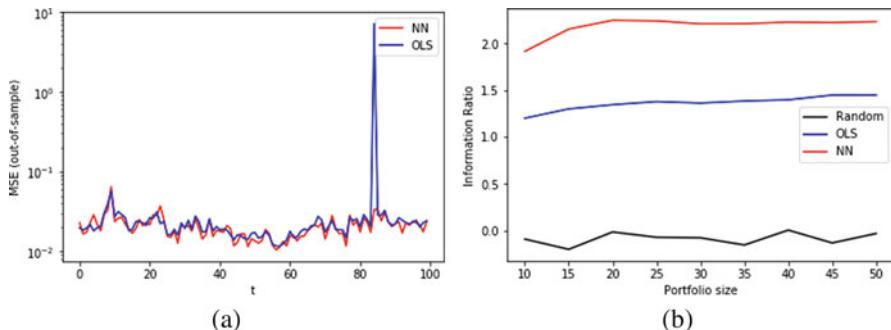


Fig. 5.10 (a) The out-of-sample MSE is compared between OLS and a two-hidden layer deep network applied to a universe of 3290 stocks from the Russell 3000 index over the period from November 2008 to November 2018. (b) The information ratios of a portfolio selection strategy which selects the n stocks from the universe with the highest predicted monthly returns. The information ratios are evaluated for various size portfolios. The information ratios are based on out-of-sample predicted asset returns using OLS regression, neural networks, and randomized selection with no predictive model

Finally, Fig. 5.11 compares the distribution of sensitivities to each factor over the entire 100 month period using the neural network (top) and OLS regression (bottom). The sensitivities are sorted in ascending order from left to right by their median values. We observe that the NN ranking of the factors differs substantially from the OLS regression.

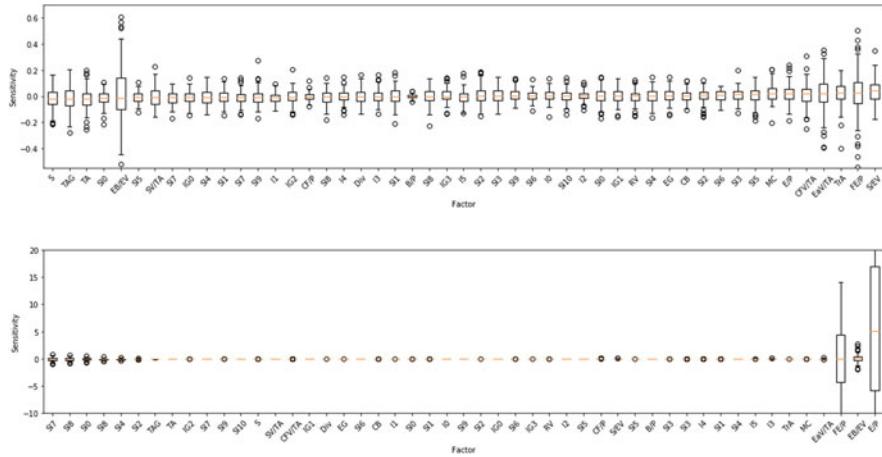


Fig. 5.11 The distribution of factor model sensitivities to each factor over the entire ten-year period using the neural network applied to the Russell 3000 asset factor loadings (top). The sensitivities are sorted in ascending order from left to right by their median values. The same sensitivities using OLS linear regression (bottom). See Table 5.4 for a short description of the fundamental factors

7 Summary

An important aspect in adoption of neural networks in factor modeling is the existence of a statistical framework which provides the transparency and statistical interpretability of linear least squares estimation. Moreover, one should expect to use such a framework applied to linear data and obtain similar results to linear regression, thus isolating the effects of non-linearity versus the effect of using different optimization algorithms and model implementation environments.

In this chapter, we introduce a deep learning framework for interpretable cross-sectional modeling and demonstrate its application to a simple fundamental factor model. Deep learning generalizes the linear fundamental factor models by capturing non-linearity, interaction effects, and non-parametric shocks in financial econometrics. This framework provides interpretability, with confidence intervals, and ranking of the factor importance and interaction effects. In the case when the network contains no hidden layers, our approach recovers a linear fundamental factor model. The framework allows the impact of non-linearity and non-parametric treatment of the error on the factors over time and forms the basis for generalized interpretability of fundamental factors.

8 Exercises

Exercise 5.1*

Consider the following data generation process

$$Y = X_1 + X_2 + \epsilon, \quad X_1, X_2, \epsilon \sim N(0, 1),$$

i.e. $\beta_0 = 0$ and $\beta_1 = \beta_2 = 1$.

- a. For this data, write down the mathematical expression for the sensitivities of the fitted neural network when the network has

- zero hidden layers;
- one hidden layer, with n unactivated hidden units;
- one hidden layer, with n tanh activated hidden units;
- one hidden layer, with n ReLU activated hidden units; and
- two hidden layers, each with n tanh activated hidden units.

Exercise 5.2**

Consider the following data generation process

$$Y = X_1 + X_2 + X_1 X_2 + \epsilon, \quad X_1, X_2 \sim N(0, 1), \quad \epsilon \sim N(0, \sigma_n^2),$$

i.e. $\beta_0 = 0$ and $\beta_1 = \beta_2 = \beta_{12} = 1$, where β_{12} is the interaction term. σ_n^2 is the variance of the noise and $\sigma_n = 0.01$.

- a. For this data, write down the mathematical expression for the interaction term (i.e., the off-diagonal components of the Hessian matrix) of the fitted neural network when the network has

- zero hidden layers;
- one hidden layer, with n unactivated hidden units;
- one hidden layer, with n tanh activated hidden units;
- one hidden layer, with n ReLU activated hidden units; and
- two hidden layers, each with n tanh activated hidden units.

Why is the ReLU activated network problematic for estimating interaction terms?

8.1 Programming Related Questions*

Exercise 5.3*

For the same problem in the previous exercise, use 5000 simulations to generate a regression training set dataset for the neural network with one hidden layer. Produce a table showing how the mean and standard deviation of the sensitivities β_i behave as the number of hidden units is increased. Compare your result with *tanh* and *ReLU* activation. What do you conclude about which

activation function to use for interpretability? Note that you should use the notebook Deep-Learning-Interpretability.ipynb as the starting point for experimental analysis.

Exercise 5.4*

Generalize the `sensitivities` function in Exercise 5.3 to L layers for either tanh or ReLU activated hidden layers. Test your function on the data generation process given in Exercise 5.1.

Exercise 5.5**

Fixing the total number of hidden units, how do the mean and standard deviation of the sensitivities β_i behave as the number of layers is increased? Your answer should compare using either tanh or ReLU activation functions. Note, do not mix the type of activation functions across layers. What do you conclude about the effect of the number of layers, keeping the total number of units fixed, on the interpretability of the sensitivities?

Exercise 5.6**

For the same data generation process as the previous exercise, use 5000 simulations to generate a regression training set for the neural network with one hidden layer. Produce a table showing how the mean and standard deviation of the interaction term behave as the number of hidden units is increased, fixing all other parameters. What do you conclude about the effect of the number of hidden units on the interpretability of the interaction term? Note that you should use the notebook Deep-Learning-Interaction.ipynb as the starting point for experimental analysis.

Appendix

Other Interpretability Methods

Partial Dependence Plots (PDPs) evaluate the expected output w.r.t. the marginal density function of each input variable, and allow the importance of the predictors to be ranked. More precisely, partitioning the data X into an *interest* set, X_s , and its complement, $X_c = X \setminus X_s$, then the “partial dependence” of the response on X_s is defined as

$$f_s(X_s) = E_{X_c} [\hat{f}(X_s, X_c)] = \int \hat{f}(X_s, X_c) p_c(X_c) dX_c, \quad (5.20)$$

where $p_c(X_c)$ is the marginal probability density of X_c : $p_c(X_c) = \int p(x) dx_s$. Equation (5.20) can be estimated from a set of training data by

$$\bar{f}_s(X_s) = \frac{1}{n} \sum_{i=1}^n \hat{f}(X_s, X_{i,c}), \quad (5.21)$$

where $X_{i,c}$ ($i = 1, 2, \dots, n$) are the observations of X_c in the training set; that is, the effects of all the other predictors in the model are averaged out. There are a number of challenges with using PDPs for model interpretability. First, the interaction effects are ignored by the simplest version of this approach. While Greenwell et al. (2018) propose a methodology extension to potentially address the modeling of interactive effects, PDPs do not provide a 1-to-1 correspondence with the coefficients in a linear regression. Instead, we would like to know, under strict control conditions, how the fitted weights and biases of the MLP correspond to the fitted coefficients of linear regression. Moreover in the context of neural networks, by treating the model as a black-box, it is difficult to gain theoretical insight into how the choice of the network architecture affects its interpretability from a probabilistic perspective.

Garson (1991) partitions hidden-output connection weights into components associated with each input neuron using absolute values of connection weights. Garson's algorithm uses the absolute values of the connection weights when calculating variable contributions, and therefore does not provide the direction of the relationship between the input and output variables.

Olden and Jackson (2002) determines the relative importance, $r_{ij} = [R]_{ij}$, of the i th output to the j th predictor variable of the model as a function of the weights, according to the expression

$$r_{ij} = W_{jk}^{(2)} W_{ki}^{(1)}. \quad (5.22)$$

The approach does not account for non-linearity introduced into the activation, which is the most critical aspects of the model. Furthermore, the approach presented was limited to a single hidden layer.

Proof of Variance Bound on Jacobian

Proof The Jacobian can be written in matrix element form as

$$J_{ij} = [\partial_X \hat{Y}]_{ij} = \sum_{k=1}^n w_{ik}^{(2)} w_{kj}^{(1)} H(I_k^{(1)}) = \sum_{k=1}^n c_k H_k(I) \quad (5.23)$$

where $c_k := c_{ijk} := w_{ik}^{(2)} w_{kj}^{(1)}$ and $H_k(I) := H(I_k^{(1)})$ is the Heaviside function. As a linear combination of indicator functions, we have

$$J_{ij} = \sum_{k=1}^{n-1} a_k \mathbb{1}_{\{I_k^{(1)} > 0, I_{k+1}^{(1)} \leq 0\}} + a_n \mathbb{1}_{\{I_n^{(1)} > 0\}}, \quad a_k := \sum_{i=1}^k c_i. \quad (5.24)$$

Alternatively, the Jacobian can be expressed in terms of a weighted sum of independent Bernoulli trials involving X :

$$J_{ij} = \sum_{k=1}^{n-1} a_k \mathbb{1}_{\{\mathbf{w}_k^{(1)} X > -b_k^{(1)}, \mathbf{w}_{k+1}^{(1)} X \leq -b_{k+1}^{(1)}\}} + a_n \mathbb{1}_{\{\mathbf{w}_n^{(1)} X > -b_n^{(1)}\}}. \quad (5.25)$$

Without loss of generality, consider the case when $p = 1$, the dimension of the input space is one. Then Eq. 5.25 simplifies to:

$$J_{ij} = \sum_{k=1}^{n-1} a_k \mathbb{1}_{x_k < X \leq x_{k+1}} + a_n \mathbb{1}_{x_n < X}, \quad j = 1, \quad (5.26)$$

where $x_k := -\frac{b_k^{(1)}}{W_k^{(1)}}$. The expectation of the Jacobian is given by

$$\mu_{ij} := \mathbb{E}[J_{ij}] = \sum_{k=1}^n a_k p_k, \quad (5.27)$$

where $p_k := \Pr(x_k < X \leq x_{k+1}) \forall k = 1, \dots, n-1$, $p_n := \Pr(x_n < X)$. For finite weights, the expectation is bounded above by $\sum_{k=1}^n a_k$. We can write the variance of the Jacobian as:

$$\mathbb{V}[J_{ij}] = \sum_{k=1}^{n-1} a_k \mathbb{V}[\mathbb{1}_{\{Z_k^{(1)} > 0, Z_{k+1}^{(1)} \leq 0\}}] + a_n \mathbb{V}[\mathbb{1}_{\{Z_n^{(1)} > 0\}}] = \sum_{k=1}^n a_k p_k (1 - p_k). \quad (5.28)$$

Under the assumption that the mean of the Jacobian is invariant to the number of hidden units, or if the weights are constrained so that the mean is constant, then the weights are $a_k = \frac{\mu_{ij}}{np_k}$. Then the variance is bounded by the mean:

$$\mathbb{V}[J_{ij}] = \mu_{ij} \frac{n-1}{n} < \mu_{ij}. \quad (5.29)$$

If we relax the assumption that μ_{ij} is independent of n then, under the original weights $a_k := \sum_{i=1}^k c_i$:

$$\begin{aligned} \mathbb{V}[J_{ij}] &= \sum_{k=1}^n a_k p_k (1 - p_k) \\ &\leq \sum_{k=1}^n a_k p_k \\ &= \mu_{ij} \\ &\leq \sum_{k=1}^n a_k. \end{aligned}$$

□

Russell 3000 Factor Model Description

Python Notebooks

The notebooks provided in the accompanying source code repository are designed to gain familiarity with how to implement interpretable deep networks. The examples include toy simulated data and a simple factor model. Further details of the notebooks are included in the README.md file.

References

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). Tensor flow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16 (pp. 265–283).
- Dimopoulos, Y., Bourret, P., & Lek, S. (1995, Dec). Use of some sensitivity criteria for choosing networks with good generalization ability. *Neural Processing Letters*, 2(6), 1–4.
- Dixon, M. F., & Polson, N. G. (2019). Deep fundamental factor models.
- Garson, G. D. (1991, April). Interpreting neural-network connection weights. *AI Expert*, 6(4), 46–51.
- Greenwell, B. M., Boehmke, B. C., & McCarthy, A. J. (2018, May). A simple and effective model-based variable importance measure. *arXiv e-prints*, arXiv:1805.04755.
- Nielsen, F., & Bender, J. (2010). The fundamentals of fundamental factor models. Technical Report 24, MSCI Barra Research Paper.
- Olden, J. D., & Jackson, D. A. (2002). Illuminating the “black box”: a randomization approach for understanding variable contributions in artificial neural networks. *Ecological Modelling*, 154(1), 135–150.
- Rosenberg, B., & Marathe, V. (1976). Common factors in security returns: Microeconomic determinants and macroeconomic correlates. Research Program in Finance Working Papers 44, University of California at Berkeley.

Part II

Sequential Learning

Chapter 6

Sequence Modeling



This chapter provides an overview of the most important modeling concepts in financial econometrics. Such methods form the conceptual basis and performance baseline for more advanced neural network architectures presented in the next chapter. In fact, each type of architecture is a generalization of many of the models presented here. This chapter is especially useful for students from an engineering or science background, with little exposure to econometrics and time series analysis.

1 Introduction

More often in finance, the data consists of observations of a variable over time, e.g. stock prices, bond yields, etc. In such a case, the observations are not independent over time, rather observations are often strongly related to their recent histories. For this reason, the ordering of the data matters (unlike cross-sectional data). This is in contrast to most methods of machine learning which assume that the data is i.i.d. Moreover algorithms and techniques for fitting machine learning models, such as back-propagation for neural networks and cross-validation for hyperparameter tuning, must be modified for use on time series data.

“Stationarity” of the data is a further important delineation necessary to successfully apply models to time series data. If the estimated moments of the data change depending on the window of observation, then the modeling problem is much more difficult. Neural network approaches to addressing these challenges are presented in the next chapter.

An additional consideration is the *data frequency*—the frequency at which the data is observed assuming that the timestamps are uniform. In general, the frequency of the data governs the frequency of the time series model. For example, support that we seek to predict the week ahead stock price from daily historical adjusted close prices on business days. In such a case, we would build a model from daily prices

and then predict 5 daily steps ahead, rather than building a model using only weekly intervals of data.

In this chapter we shall primarily consider applications of parametric, linear, and frequentist models to uniform time series data. Such methods form the conceptual basis and performance baseline for more advanced neural network architectures presented in the next chapter. In fact, each type of architecture is a generalization of many of the models presented here. Please note that the material presented in this chapter is not intended as a substitute for a more comprehensive and rigorous treatment of econometrics, but rather to provide enough background for Chap. 8.

Chapter Objectives

By the end of this chapter, the reader should expect to accomplish the following:

- Explain and analyze linear autoregressive models;
- Understand the classical approaches to identifying, fitting, and diagnosing autoregressive models;
- Apply simple heteroscedastic regression techniques to time series data;
- Understand how exponential smoothing can be used to predict and filter time series; and
- Project multivariate time series data onto lower dimensional spaces with principal component analysis.

Note that this chapter can be skipped if the reader is already familiar with econometrics. This chapter is especially useful for students from an engineering or physical sciences background, with little exposure to econometrics and time series analysis.

2 Autoregressive Modeling

We begin by considering a single variable Y_t indexed by t to indicate the variable changes over time. This variable may depend on other variables X_t ; however, we shall simply consider the case when the dependence of Y_t is on past observations of itself—this is known as *univariate time series analysis*.

2.1 Preliminaries

Before we can build a model to predict Y_t , we recall some basic definitions and terminology, starting with a continuous time setting and then continuing thereafter solely in a discrete-time setting.

➤ Stochastic Process

A stochastic process is a sequence of random variables, indexed by continuous time: $\{Y_t\}_{t=-\infty}^{\infty}$.

➤ Time Series

A time series is a sequence of observations of a stochastic process at discrete times over a specific interval: $\{y_t\}_{t=1}^n$.

➤ Autocovariance

The j th autocovariance of a time series is $\gamma_{jt} := \mathbb{E}[(y_t - \mu_t)(y_{t-j} - \mu_{t-j})]$, where $\mu_t := \mathbb{E}[y_t]$.

➤ Covariance (Weak) Stationarity

A time series is weak (or wide-sense) covariance stationary if it has time constant mean and autocovariances of all orders:

$$\begin{aligned}\mu_t &= \mu, & \forall t \\ \gamma_{jt} &= \gamma_j, & \forall t.\end{aligned}$$

As we have seen, this implies that $\gamma_j = \gamma_{-j}$: the autocovariances depend only on the interval between observations, but not the time of the observations.

➤ Autocorrelation

The j th autocorrelation, τ_j is just the j th autocovariance divided by the variance:

$$\tau_j = \frac{\gamma_j}{\gamma_0}. \tag{6.1}$$

➤ White Noise

White noise, ϵ_t , is i.i.d. error which satisfies all three conditions:

- a. $\mathbb{E}[\epsilon_t] = 0, \forall t;$
- b. $\mathbb{V}[\epsilon_t] = \sigma^2, \forall t;$ and
- c. ϵ_t and ϵ_s are independent, $t \neq s, \forall t, s.$

Gaussian white noise just adds a normality assumption to the error. White noise error is often referred to as a “disturbance,” “shock,” or “innovation” in the financial econometrics literature.

With these definitions in place, we are now ready to define autoregressive processes. Tacit in our usage of these models is that the time series exhibits autocorrelation.¹ If this is not the case, then we would choose to use cross-sectional models seen in Part I of this book.

2.2 Autoregressive Processes

Autoregressive models are parametric time series models describing y_t as a linear combination of p past observations and white noise. They are referred to as “processes” as they are representative of random processes which are dependent on one or more past values.

➤ AR(p) Process

The p th order autoregressive process of a variable Y_t depends only on the previous values of the variable plus a white noise disturbance term

$$y_t = \mu + \sum_{i=1}^p \phi_i y_{t-i} + \epsilon_t, \quad (6.2)$$

where ϵ_t is independent of $\{y_{t-1}\}_{i=1}^p$. We refer to μ as the *drift term*. p is referred to as the *order* of the model.

Defining the polynomial function $\phi(L) := (1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p)$, where y_{t-j} is a j th lagged observation of y_t given by the *Lag operator* or *Backshift* operator, $y_{t-j} = L^j[y_j]$.

The AR(p) process can be expressed in the more compact form

$$\phi(L)[y_t] = \mu + \epsilon_t. \quad (6.3)$$

¹We shall identify statistical tests for establishing autocorrelation later in this chapter.

This compact form shall be conducive to analysis describing the properties of the $AR(p)$ process. We mention in passing that the identification of the parameter p from data, i.e. the number of lags in the model rests on the data being weakly covariance stationary.²

2.3 Stability

An important property of AR(p) processes is whether past disturbances exhibit an inclining or declining impact on the current value of y as the lag increases. For example, think of the impact of a news event about a public company on the stock price movement over the next minute versus if the same news event had occurred, say, six months in the past. One should expect that the latter is much less significant than the former.

To see this, consider the AR(1) process and write y_t in terms of the inverse of $\Phi(L)$

$$y_t = \Phi^{-1}(L)[\mu + \epsilon_t], \quad (6.4)$$

so that for an AR(1) process

$$y_t = \frac{1}{1 - \phi L} [\mu + \epsilon_t] = \sum_{j=0}^{\infty} \phi^j L^j [\mu + \epsilon_t], \quad (6.5)$$

and the infinite sum will be stable, i.e. the ϕ^j terms do not grow with j , provided that $|\phi| < 1$. Conversely, unstable AR(p) processes exhibit the counter-intuitive behavior that the error disturbance terms become increasingly influential as the lag increases. We can calculate the *Impulse Response Function* (IRF), $\frac{\partial y_t}{\partial \epsilon_{t-j}}$ $\forall j$, to characterize the influence of past disturbances. For the AR(p) model, the IRF is given by ϕ^j and hence is geometrically decaying when the model is stable.

2.4 Stationarity

Another desirable property of AR(p) models is that their autocorrelation function converges to zero as the lag increases. A sufficient condition for convergence is *stationary*. From the characteristic equation

²Statistical tests for identifying the order of the model will be discussed later in the chapter.

$$\Phi(z) = (1 - \frac{z}{\lambda_1}) \cdot (1 - \frac{z}{\lambda_2}) \cdots (1 - \frac{z}{\lambda_p}) = 0, \quad (6.6)$$

it follows that a $AR(p)$ model is strictly stationary and ergodic if all the roots lie outside the unit sphere in the complex plane \mathbb{C} . That is $|\lambda_i| > 1$, $i \in \{1, \dots, p\}$ and $|\cdot|$ is the modulus of a complex number. Note that if the characteristic equation has at least one unit root, with all other roots lying outside the unit sphere, then this is a special case of non-stationarity but not strict stationarity.

> Stationarity of Random Walk

We can show that the following random walk (zero mean AR(1) process) is not strictly stationary:

$$y_t = y_{t-1} + \epsilon_t \quad (6.7)$$

Written in compact form gives

$$\Phi(L)[y_t] = \epsilon_t, \quad \Phi(L) = 1 - L, \quad (6.8)$$

and the characteristic polynomial, $\Phi(z) = 1 - z = 0$, implies that the real root $z = 1$. Hence the root is on the unit circle and the model is a special case of non-stationarity.

Finding roots of polynomials is equivalent to finding eigenvalues. The Cayley–Hamilton theorem states that the roots of any polynomial can be found by turning it into a matrix and finding the eigenvalues.

Given the p degree polynomial³:

$$q(z) = c_0 + c_1 z + \dots + c_{p-1} z^{p-1} + z^p, \quad (6.9)$$

we define the $p \times p$ companion matrix

$$C := \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 \\ -c_0 & -c_1 & \dots & -c_{p-2} & -c_{p-1} \end{pmatrix}, \quad (6.10)$$

³Notice that the z^p coefficient is 1.

then the characteristic polynomial $\det(C - \lambda I) = q(\lambda)$, and so the eigenvalues of C are the roots of q . Note that if the polynomial does not have a unit leading coefficient, then one can just divide the polynomial by that coefficient to arrive at the form of Eq. 6.9, without changing its roots. Hence the roots of any polynomial can be found by computing the eigenvalues of a companion matrix.

The AR(p) has a characteristic polynomial of the form

$$\Phi(z) = 1 - \phi_1 z - \cdots - \phi_p z^p \quad (6.11)$$

and dividing by $-\phi_p$ gives

$$q(z) = -\frac{\Phi(z)}{\phi_p} = -\frac{1}{\phi_p} + \frac{\phi_1}{\phi_p} z + \cdots + z^p \quad (6.12)$$

and hence the companion matrix is of the form

$$C := \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 \\ \frac{1}{\phi_p} - \frac{\phi_1}{\phi_p} & \dots & -\frac{\phi_{p-1}}{\phi_p} & -\frac{\phi_{p-1}}{\phi_p} & \end{pmatrix}. \quad (6.13)$$

2.5 Partial Autocorrelations

Autoregressive models carry a signature which allows its order, p , to be determined from time series data provided the data is stationary. This signature encodes the memory in the model and is given by “partial autocorrelations.” Informally each partial autocorrelation measures the correlation of a random variable, y_t , with its lag, y_{t-h} , while controlling for intermediate lags. The formal definition of the partial autocorrelation is now given.

► Partial Autocorrelation

A partial autocorrelation at lag $h \geq 2$ is a conditional autocorrelation between a variable, y_t , and its h th lag, y_{t-h} under the assumption that the values of the intermediate lags, $y_{t-1}, \dots, y_{t-h+1}$ are controlled:

$$\tilde{\tau}_h := \tilde{\tau}_{t,t-h} := \frac{\tilde{\gamma}_h}{\sqrt{\tilde{\gamma}_{t,h}} \sqrt{\tilde{\gamma}_{t-h,h}}},$$

where $\tilde{\gamma}_h := \tilde{\gamma}_{t,t-h} := \mathbb{E}[y_t - P(y_t | y_{t-1}, \dots, y_{t-h+1}), y_{t-h} - P(y_{t-h} | y_{t-1}, \dots, y_{t-h+1})]$ is the lag- h partial autocovariance, $P(W | Z)$ is an orthogonal projection of W onto the set Z and

$$\tilde{\gamma}_{t,h} := \mathbb{E}[(y_t - P(y_t | y_{t-1}, \dots, y_{t-h+1}))^2]. \quad (6.14)$$

The partial autocorrelation function $\tilde{\tau}_h : \mathbb{N} \rightarrow [-1, 1]$ is a map $h \mapsto \tilde{\tau}_h$. The plot of $\tilde{\tau}_h$ against h is referred to as the partial *correlogram*.

AR(p) Processes

Using the property that a linear orthogonal projection $\hat{y}_t = P(y_t | y_{t-1}, \dots, y_{t-h+1})$ is given by the OLS estimator as $\hat{y}_t = \phi_1 y_{t-1} + \dots + \phi_{h-1} y_{t-h+1}$, gives the *Yule-Walker equations* for an AR(p) process, relating the partial autocorrelations $\tilde{\tau}_p := [\tilde{\tau}_1, \dots, \tilde{\tau}_p]$ to the autocorrelations $\mathcal{T}_p := [\tau_1, \dots, \tau_p]$:

$$R_p \tilde{\mathcal{T}}_p = \mathcal{T}_p, \quad R_p = \begin{bmatrix} 1 & \tau_1 & \dots & \tau_{p-1} \\ \tau_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \tau_{p-1} & \tau_{p-2} & \dots & 1 \end{bmatrix}. \quad (6.15)$$

For $h \leq p$, we can solve for the h th lag partial autocorrelation by writing

$$\tilde{\tau}_h = \frac{|R_h^*|}{|R_h|}, \quad (6.16)$$

where $|\cdot|$ is the matrix determinant and the j th column of $[R_h^*]_{:,j} = [R_h]_{1,j}, j \neq h$ and the h th column is $[R_h^*]_{:,h} = \mathcal{T}_h$.

For example, the lag-1 partial autocorrelation is $\tilde{\tau}_1 = \tau_1$ and the lag-2 partial autocorrelation is

$$\tilde{\tau}_2 = \frac{\begin{vmatrix} 1 & \tau_1 \\ \tau_1 & \tau_2 \end{vmatrix}}{\begin{vmatrix} 1 & \tau_1 \\ \tau_1 & 1 \end{vmatrix}} = \frac{\tau_2 - \tau_1^2}{1 - \tau_1^2}. \quad (6.17)$$

We note, in particular, that the lag-2 partial autocorrelation of an AR(1) process, with autocorrelation $\mathcal{T}_2 = [\tau_1, \tau_1^2]$ is

$$\tilde{\tau}_2 = \frac{\tau_1^2 - \tau_1^2}{1 - \tau_1^2} = 0, \quad (6.18)$$

and this is true for all lag orders greater than the order of the AR process. We can reason about this property from another perspective—through the partial autocovariances. The lag-2 partial autocovariance of an AR(1) process is

$$\tilde{\gamma}_2 := \tilde{\gamma}_{t,t-2} := \mathbb{E}[y_t - \hat{y}_t, y_{t-2} - \hat{y}_{t-2}], \quad (6.19)$$

where $\hat{y} = P(y_t | y_{t-1})$ and $\hat{y}_{t-2} = P(y_{t-2} | y_{t-1})$. When P is a linear orthogonal projection, we have from the property of an orthogonal projection

$$P(W | Z) = \mu_W + \frac{\text{Cov}(W, Z)}{\mathbb{V}[Z]}(Z - \mu_Z) \quad (6.20)$$

and $P(y_t | y_{t-1}) = \phi \frac{\mathbb{V}(y_{t-1})}{\mathbb{V}(y_{t-1})} = \phi$ so that

$\hat{y}_t = \phi y_{t-1}$, $\hat{y}_{t-2} = \phi y_{t-1}$ and hence $\epsilon_t = y_t - \hat{y}_t$ so the lag-2 partial autocovariance is

$$\tilde{\gamma}_2 = \mathbb{E}[\epsilon_t, y_{t-2} - \phi y_{t-1}] = 0. \quad (6.21)$$

Clearly the lag-1 partial autocovariance of an AR(1) process is

$$\tilde{\gamma}_1 = \mathbb{E}[y_t - \mu, y_{t-1} - \mu] = \gamma_1 = \phi \gamma_0. \quad (6.22)$$

2.6 Maximum Likelihood Estimation

The exact likelihood when the density of the data is independent of (ϕ, σ_n^2) is

$$\mathcal{L}(y, x; \phi, \sigma_n^2) = \prod_{t=1}^T f_{Y_t|X_t}(y_t | x_t; \phi, \sigma_n^2) f_{X_t}(x_t). \quad (6.23)$$

Under this assumption, the exact likelihood is proportional to the conditional likelihood function:

$$\begin{aligned} \mathcal{L}(y, x; \phi, \sigma_n^2) &\propto L(y|x; \phi, \sigma_n^2) \\ &= \prod_{t=1}^T f_{Y_t|X_t}(y_t | x_t; \phi, \sigma_n^2) \\ &= (\sigma_n^2 2\pi)^{-T/2} \exp\left\{-\frac{1}{2\sigma_n^2} \sum_{t=1}^T (y_t - \phi^T \mathbf{x}_t)^2\right\}. \end{aligned}$$

In many cases such an assumption about the independence of the density of the data and the parameters is not warranted. For example, consider the zero mean AR(1) with unknown noise variance:

$$y_t = \phi y_{t-1} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_n^2) \quad (6.24)$$

$$Y_t | Y_{t-1} \sim \mathcal{N}(\phi y_{t-1}, \sigma_n^2)$$

$$Y_1 \sim \mathcal{N}(0, \frac{\sigma_n^2}{1 - \phi^2}).$$

The exact likelihood is

$$\begin{aligned} \mathcal{L}(x; \phi, \sigma_n^2) &= f_{Y_t|Y_{t-1}}(y_t|y_{t-1}; \phi, \sigma_n^2) f_{Y_1}(y_1; \phi, \sigma_n^2) \\ &= \left(\frac{\sigma_n^2}{1 - \phi^2} 2\pi \right)^{-1/2} \exp\left\{-\frac{1 - \phi^2}{2\sigma_n^2} y_1^2\right\} (\sigma_n^2 2\pi)^{-\frac{T-1}{2}} \\ &\quad \exp\left\{-\frac{1}{2\sigma_n^2} \sum_{t=2}^T (y_t - \phi y_{t-1})^2\right\}, \end{aligned}$$

where we made use of the moments of Y_t —a result which is derived in Sect. 2.8.

Despite the dependence of the density of the data on the parameters, there may be practically little advantage of using exact maximum likelihood against the conditional likelihood method (i.e., dropping the $f_{Y_1}(y_1; \phi, \sigma_n^2)$ term). This turns out to be the case for linear models. Maximizing the conditional likelihood is equivalent to ordinary least squares estimation.

2.7 Heteroscedasticity

The AR model assumes that the noise is i.i.d. This may be an overly optimistic assumption which can be relaxed by assuming that the noise is time dependent. Treating the noise as time dependent is exemplified by a heteroscedastic AR(p) model

$$y_t = \mu + \sum_{i=1}^p \phi_i y_{t-i} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_{n,t}^2). \quad (6.25)$$

There are many tests for heteroscedasticity in time series models and one of them, the ARCH test, is summarized in Table 6.3. The estimation procedure for heteroscedastic models is more complex and involves two steps: (i) estimation of the errors from the maximum likelihood function which treats the errors as independent and (ii) estimation of model parameters under a more general maximum likelihood

estimation which treats the errors as time-dependent. Note that such a procedure could be generalized further to account for correlation in the errors but requires the inversion of the covariance matrix, which is computationally intractable with large time series.

The conditional likelihood is

$$\begin{aligned}\mathcal{L}(\mathbf{y}|X; \phi, \sigma_n^2) &= \prod_{t=1}^T f_{Y_t|X_t}(y_t|x_t; \phi, \sigma_n^2) \\ &= (2\pi)^{-T/2} \det(D)^{-1/2} \exp\left\{-\frac{1}{2}(\mathbf{y} - \phi^T X)^T D^{-1}(\mathbf{y} - \phi^T X)\right\},\end{aligned}$$

where $D_{tt} = \sigma_{n,t}^2$ is the diagonal covariance matrix and $X \in \mathbb{R}^{T \times p}$ is the data matrix defined as $[X]_t = \mathbf{x}_t$.

The advantage of this approach is its relative simplicity. The treatment of noise variance as time dependent in finance has long been addressed by more sophisticated econometrics models and the approach presented here brings AR models into line with the specifications of more realistic models.

On the other hand, the use of the sample variance of the residuals is only appropriate when the sample size is sufficient. In practice, this translates into the requirement for a sufficiently large historical period before a prediction can be made. Another disadvantage is that the approach does not explicitly define the relationship between the variances. We shall briefly revisit heteroscedastic models and explore a model for regressing the conditional variance on previous conditional variances in Sect. 2.9.

2.8 Moving Average Processes

The Wold representation theorem (a.k.a. Wold decomposition) states that every covariance stationary time series can be written as the sum of two time series, one deterministic and one stochastic. In effect, we have already considered the deterministic component when choosing an AR process.⁴ The stochastic component can be represented as a “moving average process” or MA(q) process which expresses y_t as a linear combination of current and q past disturbances. Its definition is as follows:

MA(q) Process

The q th order moving average process is the linear combination of the white noise process $\{\epsilon_{t-i}\}_{t=0}^q$, $\forall t$

⁴This is an overly simplistic statement because the AR(1) process can be expressed as a MA(∞) process and vice versa.

$$y_t = \mu + \sum_{i=1}^q \theta_i \epsilon_{t-i} + \epsilon_t. \quad (6.26)$$

It turns out that y_{t-1} depends on $\{\epsilon_{t-1}, \epsilon_{t-2}, \dots\}$, but not ϵ_t and hence $\gamma_{t,t-2}^2 = 0$. It should be apparent that this property holds even when P is a non-linear projection provided that the errors are independent (but not necessarily identical).

Another brief point of discussion is that an AR(1) process can be rewritten as a MA(∞) process. Suppose that the AR(1) process has a mean μ and the variance of the noise is σ_n^2 , then by a binomial expansion of the operator $(1 - \phi L)^{-1}$ we have

$$y_t = \frac{\mu}{1 - \phi} + \sum_{j=0}^{\infty} \phi^j \epsilon_{t-j}, \quad (6.27)$$

where the moments can be easily found and are

$$\begin{aligned} \mathbb{E}[y_t] &= \frac{\mu}{1 - \phi} \\ \mathbb{V}[y_t] &= \sum_{j=0}^{\infty} \phi^{2j} \mathbb{E}[\epsilon_{t-j}^2] \\ &= \sigma_n^2 \sum_{j=0}^{\infty} \phi^{2j} = \frac{\sigma_n^2}{1 - \phi^2}. \end{aligned}$$

AR and MA models are important components of more complex models which are known as ARMA or, more generally, ARIMA models. The expression of a pattern as a linear combination of past observations and past innovations turns out to be more flexible in time series modeling than any single component. These are by no means the only useful techniques and we briefly turn to another technique which smooths out shorter-term fluctuations and consequently boosts the signal to noise ratio in longer term predictions.

2.9 GARCH

Recall from Sect. 2.7 that heteroscedastic time series models treat the error as time dependent. A popular parametric, linear, and heteroscedastic method used in financial econometrics is the Generalized Autoregressive Conditional Heteroscedastic (GARCH) model (Bollerslev and Taylor). A GARCH(p,q) model specifies that the conditional variance (i.e., volatility) is given by an ARMA(p,q) model—there are p lagged conditional variances and q lagged squared noise terms:

$$\sigma_t^2 := \mathbb{E}[\epsilon_t^2 | \Omega_{t-1}] = \alpha_0 + \sum_{i=1}^q \alpha_i \epsilon_{t-i}^2 + \sum_{i=1}^p \beta_i \sigma_{t-i}^2.$$

This model gives an explicit relationship between the current volatility and previous volatilities. Such a relationship is useful for predicting volatility in the model, with obvious benefits for volatility modeling in trading and risk management. This simple relationship enables us to characterize the behavior of the model, as we shall see shortly.

A necessary condition for model stationarity is the following constraint:

$$\left(\sum_{i=1}^q \alpha_i + \sum_{i=1}^p \beta_i \right) < 1.$$

When the model is stationary, the long-run volatility converges to the unconditional variance of ϵ_t :

$$\sigma^2 := \text{var}(\epsilon_t) = \frac{\alpha_0}{1 - (\sum_{i=1}^q \alpha_i + \sum_{i=1}^p \beta_i)}.$$

To see this, let us consider the l-step ahead forecast using a GARCH(1,1) model:

$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \quad (6.28)$$

$$\hat{\sigma}_{t+1}^2 = \alpha_0 + \alpha_1 \mathbb{E}[\epsilon_t^2 | \Omega_{t-1}] + \beta_1 \sigma_t^2 \quad (6.29)$$

$$= \sigma^2 + (\alpha_1 + \beta_1)(\sigma_t^2 - \sigma^2) \quad (6.30)$$

$$\hat{\sigma}_{t+2}^2 = \alpha_0 + \alpha_1 \mathbb{E}[\epsilon_{t+1}^2 | \Omega_{t-1}] + \beta_1 \mathbb{E}[\sigma_{t+1}^2 | \Omega_{t-1}] \quad (6.31)$$

$$= \sigma^2 + (\alpha_1 + \beta_1)^2(\sigma_t^2 - \sigma^2) \quad (6.32)$$

$$\hat{\sigma}_{t+l}^2 = \alpha_0 + \alpha_1 \mathbb{E}[\epsilon_{t+l-1}^2 | \Omega_{t-1}] + \beta_1 \mathbb{E}[\sigma_{t+l-1}^2 | \Omega_{t-1}] \quad (6.33)$$

$$= \sigma^2 + (\alpha_1 + \beta_1)^l(\sigma_t^2 - \sigma^2), \quad (6.34)$$

where we have substituted for the unconditional variance, $\sigma^2 = \alpha_0 / (1 - \alpha_1 - \beta_1)$. From the above equation we can see that $\hat{\sigma}_{t+1}^2 \rightarrow \sigma^2$ as $l \rightarrow \infty$ so as the forecast horizon goes to infinity, the variance forecast approaches the unconditional variance of ϵ_t . From the l-step ahead variance forecast, we can see that $(\alpha_1 + \beta_1)$ determines how quickly the variance forecast converges to the unconditional variance. If the variance sharply rises during a crisis, the number of periods, K , until it is halfway between the first forecast and the unconditional variance is $(\alpha_1 + \beta_1)^K = 0.5$, so the half-life⁵ is given by $K = \ln(0.5) / \ln(\alpha_1 + \beta_1)$.

⁵The half-life is the lag k at which its coefficient is equal to a half.

For example, if

$$(\alpha_1 + \beta_1) = 0.97$$

and steps are measured in days, the half-life is approximately 23 days.

2.10 Exponential Smoothing

Exponential smoothing is a type of forecasting or filtering method that exponentially decreases the weight of past and current observations to give smoothed predictions \tilde{y}_{t+1} . It requires a single parameter, α , also called the smoothing factor or smoothing coefficient. This parameter controls the rate at which the influence of the observations at prior time steps decay exponentially. α is often set to a value between 0 and 1. Large values mean that the model pays attention mainly to the most recent past observations, whereas smaller values mean more of the history is taken into account when making a prediction. Exponential smoothing takes the forecast for the previous period \tilde{y}_t and adjusts with the forecast error, $y_t - \tilde{y}_t$. The forecast for the next period becomes

$$\tilde{y}_{t+1} = \tilde{y}_t + \alpha(y_t - \tilde{y}_t), \quad (6.35)$$

or equivalently

$$\tilde{y}_{t+1} = \alpha y_t + (1 - \alpha)\tilde{y}_t. \quad (6.36)$$

Writing this as a geometric decaying autoregressive series back to the first observation:

$$\begin{aligned} \tilde{y}_{t+1} &= \alpha y_t + \alpha(1 - \alpha)y_{t-1} + \alpha(1 - \alpha)^2 y_{t-2} + \alpha(1 - \alpha)^3 y_{t-3} \\ &\quad + \cdots + \alpha(1 - \alpha)^{t-1} y_1 + \alpha(1 - \alpha)^t \tilde{y}_1, \end{aligned}$$

hence we observe that smoothing introduces a long-term model of the entire observed data, not just a sub-sequence used for prediction in a AR model, for example. For geometrically decaying models, it is useful to characterize it by the half-life—the lag k at which its coefficient is equal to a half:

$$\alpha(1 - \alpha)^k = \frac{1}{2}, \quad (6.37)$$

or

$$k = -\frac{\ln(2\alpha)}{\ln(1 - \alpha)}. \quad (6.38)$$

The optimal amount of smoothing, $\hat{\alpha}$, is found by maximizing a likelihood function.

3 Fitting Time Series Models: The Box–Jenkins Approach

While maximum likelihood estimation is the approach of choice for fitting the ARMA models described in this chapter, there are many considerations beyond fitting the model parameters. In particular, we know from earlier chapters that the bias–variance tradeoff is a central consideration which is not addressed in maximum likelihood estimation without adding a penalty term.

Machine learning achieves generalized performance through optimizing the bias–variance tradeoff, with many of the parameters being optimized through cross-validation. This is both a blessing and a curse. On the one hand, the heavy reliance on numerical optimization provides substantial flexibility but at the expense of computational cost and, often-times, under-exploitation of structure in the time series. There are also potential instabilities whereby small changes in hyperparameters lead to substantial differences in model performance.

If one were able to restrict the class of functions represented by the model, using knowledge of the relationship and dependencies between variables, then one could in principle reduce the complexity and improve the stability of the fitting procedure.

For some 75 years, econometricians and statisticians have approached the problem of time series modeling with ARIMA in a simple and intuitive way. They follow a three-step process to fit and assess $AR(p)$. This process is referred to as a Box–Jenkins approach or framework. The three basic steps of the Box–Jenkins modeling approach are:

- a. (I)dentification—determining the order of the model (a.k.a. model selection);
- b. (E)stimation—estimation of model parameters; and
- c. (D)iagnostic checking—evaluating the fit of the model.

This modeling approach is iterative and *parsimonious*—it favors models with fewer parameters.

3.1 Stationarity

Before the order of the model can be determined, the time series must be tested for stationarity. A standard statistical test for covariance stationarity is the Augmented Dickey–Fuller (ADF) test which often accounts for the (c)onstant drift and (t)ime trend. The ADF test is a unit root test—the Null hypothesis is that the characteristic polynomial exhibits at least a unit root and hence the data is non-stationary. If the Null can be rejected at a confidence level, α , then the data is stationary. Attempting to fit a time series model to non-stationary data will result in dubious interpretations of the estimated partial autocorrelation function and poor predictions and should therefore be avoided.

3.2 Transformation to Ensure Stationarity

Any trending time series process is non-stationary. Before we can fit an AR(p) model, it is first necessary to transform the original time series into a stationary form. In some instances, it may be possible to simply detrend the time series (a transformation which works in a limited number of cases). However this is rarely full proof. To the potential detriment of the predictive accuracy of the model, we can however systematically difference the original time series one or more times until we arrive at a stationary time series.

To gain insight, let us consider a simple example. Suppose we are given the following linear model with a time trend of the form:

$$y_t = \alpha + \beta t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, 1). \quad (6.39)$$

We first observe that the mean of y_t is time dependent:

$$\mathbb{E}[y_t] = \alpha + \beta t, \quad (6.40)$$

and thus this model is non-stationary. Instead we can difference the process to give

$$y_t - y_{t-1} = (\alpha + \beta t + \epsilon_t) - (\alpha + \beta(t-1) + \epsilon_{t-1}) = \beta + \epsilon_t - \epsilon_{t-1}, \quad (6.41)$$

and hence the mean and the variance of this difference process are constant and the difference process is stationary:

$$\mathbb{E}[y_t - y_{t-1}] = \beta \quad (6.42)$$

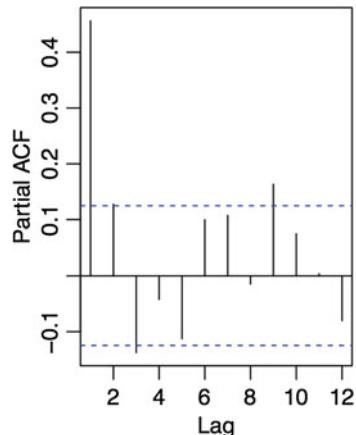
$$\mathbb{E}[(y_t - y_{t-1} - \beta)^2] = 2\sigma^2. \quad (6.43)$$

Any difference process can be written as an ARIMA(p,d,q) process, where here $d = 1$ is the order of differencing to achieve stationarity. There is, in general, no guarantee that first order differencing yields a stationary difference process. One can apply higher order differencing, $d > 1$, to the detriment of recovering the original signal, but one must often resort to non-stationary time series methods such as Kalman filters, Markov-switching models, and advanced neural networks for sequential data covered later in this part of the book.

3.3 Identification

A common approach for determining the order of a $AR(p)$ from a stationary time series is to estimate the partial autocorrelations and determine the largest lag which is significant. Figure 6.1 shows the *partial correlogram*, the plot of the estimated

Fig. 6.1 This plot shows the partial correlogram, the plot of the estimated partial autocorrelations against the lag. The solid horizontal lines define the 95% confidence interval. We observe that all but the first lag are approximately within the envelope and hence we may determine the order of the AR(p) model as $p = 1$



partial autocorrelations against the lag. The solid horizontal lines define the 95% confidence interval which can be constructed for each coefficient using

$$\pm 1.96 \times \frac{1}{\sqrt{T}}, \quad (6.44)$$

where T is the number of observations. Note that we have assumed that T is sufficiently large that the autocorrelation coefficients are assumed to be normally distributed with zero mean and standard error of $\frac{1}{\sqrt{T}}$.⁶

We observe that all but the first lag are approximately within the envelope and hence we may determine the order of the AR(p) model as $p = 1$.

The properties of the partial autocorrelation and autocorrelation plots reveal the orders of the AR and MA models. In Fig. 6.1, there is an immediate cut-off in the partial autocorrelation (acf) plot after 1 lag indicating an AR(1) process. Conversely, the location of a sharp cut-off in the estimated autocorrelation function determines the order, q , of a MA process. It is often assumed that the data generation process is a combination of an AR and MA model—referred to as an ARMA(p,q) model.

Information Criterion

While the partial autocorrelation function is useful for determining the AR(p) model order, in many cases there is an undesirable element of subjectively in the choice.

It is often preferable to use the Akaike Information Criteria (AIC) to measure the quality of fit. The AIC is given by

$$AIC = \ln(\hat{\sigma}^2) + \frac{2k}{T}, \quad (6.45)$$

⁶This assumption is admitted by the Central Limit Theorem.

where $\hat{\sigma}^2$ is the residual variance (the residual sums of squares divided by the number of observations T) and $k = p + q + 1$ is the total number of parameters estimated. This criterion expresses a bias–variance tradeoff between the first term, the quality of fit, and the second term, a penalty function proportional to the number of parameters. The goal is to select the model which minimizes the AIC by first using maximum likelihood estimation and then adding the penalty term. Adding more parameters to the model reduces the residuals but increases the right-hand term, thus the AIC favors the best fit with the fewest number of parameters.

On the surface, the overall approach has many similarities with regularization in machine learning where the loss function is penalized by a LASSO penalty (L_1 norm of the parameters) or a ridge penalty (L_2 norm of the parameters). However, we emphasize that AIC is *estimated post-hoc*, once the maximum likelihood function is evaluated, whereas in machine learning models, the penalized loss function is directly minimized.

3.4 Model Diagnostics

Once the model is fitted we must assess whether the residual exhibits autocorrelation, suggesting the model is underfitting. The residual of fitted time series model should be white noise. To test for autocorrelation in the residual, Box and Pierce propose the Portmanteau statistic

$$Q^*(m) = T \sum_{l=1}^m \hat{\rho}_l^2,$$

as a test statistic for the Null hypothesis

$$H_0 : \hat{\rho}_1 = \dots = \hat{\rho}_m = 0$$

against the alternative hypothesis

$$H_a : \hat{\rho}_i \neq 0$$

for some $i \in \{1, \dots, m\}$. $\hat{\rho}_i$ are the sample autocorrelations of the residual.

The Box-Pierce statistic follows an asymptotically chi-squared distribution with m degrees of freedom. The closely related Ljung–Box test statistic increases the power of the test in finite samples:

$$Q(m) = T(T+2) \sum_{l=1}^m \frac{\hat{\rho}_l^2}{T-l}. \quad (6.46)$$

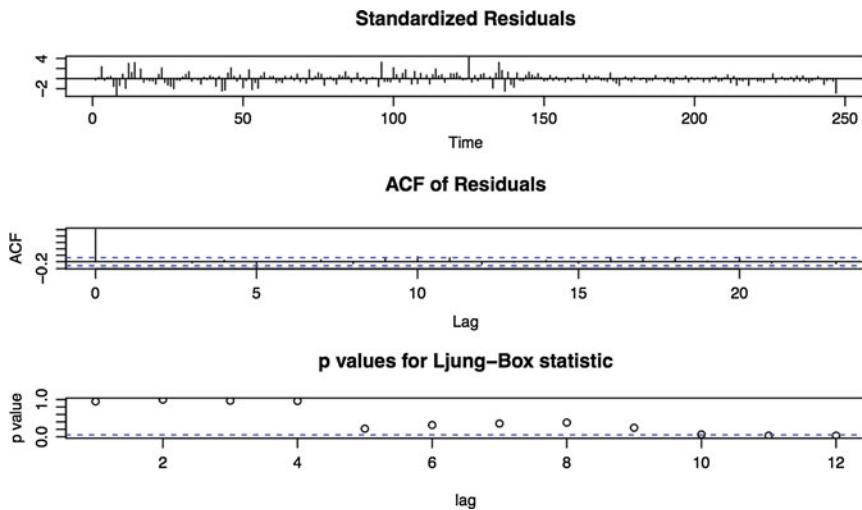


Fig. 6.2 This plot shows the results of applying a Ljung–Box test to the residuals of an AR(p) model. (Top) The standardized residuals are shown against time. (Center) The estimated ACF of the residuals is shown against the lag index. (Bottom) The p-values of the Ljung–Box test statistic are shown against the lag index

This statistic also follows asymptotically a chi-squared distribution with m degrees of freedom. The decision rule is to reject H_0 if $Q(m) > \chi_{\alpha}^2$ where χ_{α}^2 denotes the $100(1 - \alpha)$ th percentile of a chi-squared distribution with m degrees of freedom and is the significance level for rejecting H_0 .

For a $AR(p)$ model, the Ljung–Box statistic follows asymptotically a chi-squared distribution with $m - p$ degrees of freedom. Figure 6.2 shows the results of applying a Ljung–Box test to the residuals of an AR(p) model. (Top) The standardized residuals are shown against time. (Center) The estimated ACF of the residuals is shown against the lag index. (Bottom) The p-values of the Ljung–Box test statistic are shown against the lag index. The figure shows that if the maximum lag in the model is sufficiently large, then the p-value is small and the Null is rejected in favor of the alternative hypothesis.

Failing the test requires repeating the Box–Jenkins approach until the model no longer under-fits. The only mild safe-guard against over-fitting is the use of AIC for model selection, but in general there is no strong guarantee of avoiding over-fitting as the performance of the model is not assessed out-of-sample in this framework. Assessing the bias variance tradeoff by cross-validation has arisen as the better approach for generalizing model performance. Therefore any model that has been fitted under a Box–Jenkins approach needs to be assessed out-of-sample by time series cross-validation—the topic of the next section.

There are many diagnostic tests which have been developed for time series modeling which we have not discussed here. A small subset has been listed in Table 6.3. The readers should refer to a standard financial econometrics textbook

such as Tsay (2010) for further details of these tests and elaboration on their application to linear models.

4 Prediction

While the Box–Jenkins approach is useful in identifying, fitting, and critiquing models, there is no guarantee that such a model shall exhibit strong predictive properties of course. We seek to predict the value of y_{t+h} given information Ω_t up to and including time t . Producing a forecast is simply a matter of taking the conditional expectation of the data under the model. The h -step ahead forecast from a $AR(p)$ model is given by

$$\hat{y}_{t+h} = \mathbb{E}[y_{t+h} | \Omega_t] = \sum_{i=1}^p \phi_i \hat{y}_{t+h-i}, \quad (6.47)$$

where $\hat{y}_{t+h} = y_{t+h}$, $h \leq 0$ and $\mathbb{E}[y_{t+h} | \Omega_t] = 0$, $h > 0$. Note that conditional expectations of observed variables are not equal to the unconditional expectations. In particular $\mathbb{E}[y_{t+h} | \Omega_t] = \epsilon_t + h$, $h \leq 0$, whereas $\mathbb{E}[y_{t+h}] = 0$. The quality of the forecast is measured over the forecasting horizon from either the MSE or the MAE.

4.1 Predicting Events

If the output is categorical, rather than continuous, then the ARMA model is used to predict the log-odds ratio of the binary event rather than the conditional expectation of the response. This is analogous to using a logit function as a link in logistic regression.

Other general metrics are also used to assess model accuracy such as a confusion matrix, the F1-score and Receiver Operating Characteristic (ROC) curves. These metrics are not specific to time series data and could be applied to cross-section models to. The following example will illustrate a binary event prediction problem using time series data.

Example 6.1 Predicting Binary Events

Suppose we have conditionally i.i.d. Bernoulli r.v.s X_t with $p_t := \mathbb{P}(X_t = 1 | \Omega_t)$ representing a binary event and conditional moments given by

(continued)

Example 6.1 (continued)

- $\mathbb{E}[X_t | \Omega] = 0 \cdot (1 - p_t) + 1 \cdot p_t = p_t$
- $\mathbb{V}[X_t | \Omega] = p_t(1 - p_t)$

The log-odds ratio shall be assumed to follow an ARMA model,

$$\ln\left(\frac{p_t}{1 - p_t}\right) = \phi^{-1}(L)(\mu + \theta(L)\epsilon_t). \quad (6.48)$$

and the category of the model output is determined by a threshold, e.g. $p_t \geq 0.5$ corresponds to a positive event. If the number of out-of-sample observations is 24, we can compare the prediction with the observed event and construct a truth table (a.k.a. confusion matrix) as illustrated in Table 6.1.

In this example, the accuracy is $(12 + 2)/24$ —the ratio of the sum of the diagonal terms to the set size. The type I (false positive) and type II (false negative) errors, shown by the off-diagonal elements as 8 and 2, respectively.

Table 6.1 The confusion matrix for the above example

Actual	Predicted		Sum
	1	0	
1	12	2	14
0	8	2	10
Sum	20	4	24

In this example, the accuracy is $(12 + 2)/24$ —the ratio of the sum of the diagonal terms to the set size. Of special interest are the type I (false positive) and type II (false negative) errors, shown by the off-diagonal elements as 8 and 2, respectively. In practice, careful consideration must be given as to whether there is equal tolerance for type 1 and type 2 errors.

The significance of the classifier can be estimated from a chi-squared statistic with one degree of freedom under a Null hypothesis that the classifier is a white noise. In general, Chi-squared testing is used to determine whether two variables are independent of one another. In this case, if the Chi-squared statistic is above a given critical threshold value, associated with a significance level, then we can say that the classifier is not white noise.

Let us label the elements of the confusion matrix as in Table 6.2 below. The column and row sums of the confusion matrices and the total number of test samples, m , are also shown.

The chi-squared statistic with one degree of freedom is given by the squared difference of the expected result (i.e., a white noise model where the prediction is independent of the observations) and the model prediction, \hat{Y} , relative to the expected result. When normalized by the number of observations, each element of the confusion matrix is the joint probability $[\mathbb{P}(Y, \hat{Y})]_{ij}$. Under a white noise model,

Table 6.2 The confusion matrix of a binary classification is shown together with the column and row sums and the total number of test samples, m

	Predicted		
Actual	1	0	Sum
1	m_{11}	m_{12}	$m_{1,}$
0	m_{21}	m_{22}	$m_{1,}$
Sum	$m_{:,1}$	$m_{:,2}$	m

the observed outcome, Y , and the predicted outcome, \hat{Y} , are independent and so $[\mathbb{P}(Y, \hat{Y})]_{ij} = [\mathbb{P}(Y)]_i [\mathbb{P}(\hat{Y})]_j$ which is the i th row sum, $m_{i,}$, multiplied by the j th column sum, $m_{:,j}$, divided by m . Since $m_{i,j}$ is based on the model prediction, the chi-squared statistic is thus

$$\chi^2 = \sum_{i=1}^2 \sum_{j=1}^2 \frac{(m_{ij} - m_{i,} m_{:,j}/m)^2}{m_{i,} m_{:,j}/m}. \quad (6.49)$$

Returning to the example above, the chi-squared statistic is

$$\begin{aligned} \chi^2 &= (12 - (14 \times 20)/24)^2/(14 \times 20)/24 \\ &\quad + (2 - (14 \times 4)/24)^2/(14 \times 4)/24 \\ &\quad + (8 - (10 \times 20)/24)^2/(10 \times 20)/24 \\ &\quad + (2 - (10 \times 4)/24)^2/(10 \times 4)/24 \\ &= 0.231. \end{aligned}$$

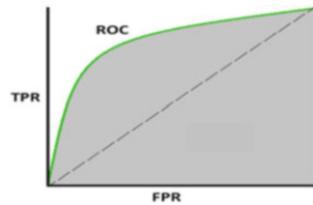
This value is far below the threshold value of 6.635 for a chi-squared statistic with one degree of freedom to be significant. Thus we cannot reject the Null hypothesis. The predicted model is not sufficiently indistinguishable from white noise.

The example classification model shown above used a threshold of $p_t >= 0.5$ to classify an event as positive. This choice of threshold is intuitive but arbitrary. How can we measure the performance of a classifier for a range of thresholds?

A ROC-Curve contains information about all possible thresholds. The ROC-Curve plots *true positive rates* against *false positive rates*, where these terms are defined as follows:

- *True Positive Rate (TPR)* is $TP/(TP + FN)$: fraction of positive samples which the classifier correctly identified. This is also known as *Recall* or *Sensitivity*. Using the confusion matrix in Table 6.1, the TPR= $12/(12 + 2) = 6/7$.
- *False Positive Rate (FPR)* is $FP/(FP + TN)$: fraction of positive samples which the classifier misidentified. In the example confusion matrix, the FPR= $8/(8 + 2) = 4/5$.
- *Precision* is $TP/(TP + FP)$: fraction of samples that were positive from the group that the classifier predicted to be positive. From the example confusion matrix, the precision is $12/(12 + 8) = 3/5$.

Fig. 6.3 The ROC curve for an example model shown by the green line



Each point in a ROC curve is a (TPR, FPR) pair for a particular choice of the threshold in the classifier. The straight dashed black line in Fig. 6.3 represents a random model. The green line shows the ROC curve of the model—importantly it is should always be above the line. The perfect model would exhibit a TPR of unity for all FPRs, so that there is no area above the curve.

The advantage of this performance measure is that it is robust to class imbalance, e.g. rare positive events. This is not true of classification accuracy which leads to misinterpretation of the quality of the fit when the data is imbalanced. For example, a constant model $\hat{Y} = f(X) = 1$ would be $x\%$ accurate if the data consists of $x\%$ positive events. Additional related metrics can be derived. Common ones include Area Under the Curve (AUC), which is the area under the green line in Fig. 6.3.

The *F1-score* is the harmonic mean of the precision and recall and is also frequently used. The F1-score reaches its best value at unity and worst at zero and is given by $F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$. From the example above $F_1 = \frac{2 \times 3 / 5 \times 6 / 7}{3 / 5 + 6 / 7} = 0.706$.

4.2 Time Series Cross-Validation

Cross-validation—the method of hyperparameter tuning by rotating through K folds (or subsets) of training-test data—differs for time series data. In prediction models over time series data, no future observations can be used in the training set. Instead, a sliding window must be used to train and predict out-of-sample over multiple repetitions to allow for parameter tuning as illustrated in Fig. 6.4. One frequent challenge is whether to fix the length of the window or allow it to “telescope” by including the ever extending history of observations as the window is “walked forward.” In general, the latter has the advantage of including more observations in the training set but can lead to difficulty in interpreting the confidence of the parameters, due to the loss of control of the sample size.

5 Principal Component Analysis

The final section in this chapter approaches data modeling from quite a different perspective, with the goal being to reduce the dimension of multivariate time series

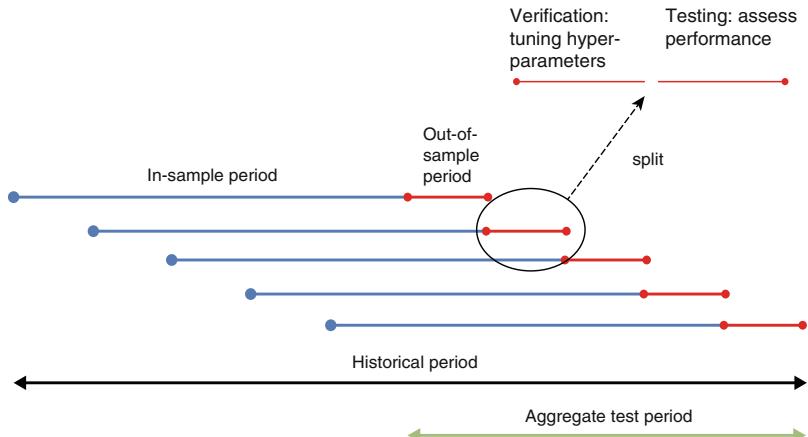


Fig. 6.4 Times series cross-validation, also referred to as “walk forward optimization,” is used instead of standard cross-validation for cross-sectional data to preserve the ordering of observations in time series data. This experimental design avoids look-ahead bias in the fitted model which occurs when one or more observations in the training set are from the future

data. The approach is widely used in finance, especially when the dimensionality of the data presents barriers to computational tractability or practical risk management and trading challenges such as hedging exposure to market risk factors. For example, it may be advantageous to monitor a few risk factors in a large portfolio rather than each instrument. Moreover such factors should provide economic insight into the behavior of the financial markets and be actionable from an investment management perspective.

Formally, let $\{\mathbf{y}_i\}_{i=1}^N$ be a set of N observation vectors, each of dimension n . We assume that $n \leq N$. Let $\mathbf{Y} \in \mathbb{R}^{n \times N}$ be a matrix whose columns are $\{\mathbf{y}_i\}_{i=1}^N$,

$$\mathbf{Y} = \begin{bmatrix} & & \\ | & & | \\ \mathbf{y}_1 & \cdots & \mathbf{y}_N \\ | & & | \end{bmatrix}.$$

The element-wise average of the N observations is an n dimensional signal which may be written as:

$$\bar{\mathbf{y}} = \frac{1}{N} \sum_{i=1}^N \mathbf{y}_i = \frac{1}{N} \mathbf{Y} \mathbf{1}_N,$$

where $\mathbf{1}_N \in \mathbb{R}^{N \times 1}$ is a column vector of all-ones. Denote \mathbf{Y}_0 as a matrix whose columns are the demeaned observations (we center each observation \mathbf{y}_i by subtracting $\bar{\mathbf{y}}$ from it):

$$\mathbf{Y}_0 = \mathbf{Y} - \bar{\mathbf{y}}\mathbb{1}_N^T.$$

Projection

A linear projection from \mathbb{R}^m to \mathbb{R}^n is a linear transformation of a finite dimensional vector given by a matrix multiplication:

$$\mathbf{x}_i = \mathbf{W}^T \mathbf{y}_i,$$

where $\mathbf{y}_i \in \mathbb{R}^n$, $\mathbf{x}_i \in \mathbb{R}^m$, and $\mathbf{W} \in \mathbb{R}^{n \times m}$. Each element j in the vector \mathbf{x}_i is an inner product between \mathbf{y}_i and the j -th column of \mathbf{W} , which we denote by \mathbf{w}_j .

Let $\mathbf{X} \in \mathbb{R}^{m \times N}$ be a matrix whose columns are the set of N vectors of transformed observations, let $\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i = \frac{1}{N} \mathbf{X} \mathbb{1}_N$ be the element-wise average, and $\mathbf{X}_0 = \mathbf{X} - \bar{\mathbf{x}} \mathbb{1}_N^T$ the demeaned matrix. Clearly, $\mathbf{X} = \mathbf{W}^T \mathbf{Y}$ and $\mathbf{X}_0 = \mathbf{W}^T \mathbf{Y}_0$.

5.1 Principal Component Projection

When the matrix \mathbf{W}^T represents the transformation that applies principal component analysis, we denote $\mathbf{W} = \mathbf{P}$, and the columns of the orthonormal matrix,⁷ \mathbf{P} , denoted $\{\mathbf{p}_j\}_{j=1}^n$, are referred to as *loading vectors*. The transformed vectors $\{\mathbf{x}_i\}_{i=1}^N$ are referred to as *principal components* or *scores*.

The first loading vector is defined as the unit vector with which the inner products of the observations have the greatest variance:

$$\mathbf{p}_1 = \max_{\mathbf{w}_1} \mathbf{w}_1^T \mathbf{Y}_0 \mathbf{Y}_0^T \mathbf{w}_1 \text{ s.t. } \mathbf{w}_1^T \mathbf{w}_1 = 1. \quad (6.50)$$

The solution to Eq. 6.50 is known to be the eigenvector of the sample covariance matrix $\mathbf{Y}_0 \mathbf{Y}_0^T$ corresponding to its largest eigenvalue.⁸

Next, \mathbf{p}_2 is the unit vector which has the largest variance of inner products between it and the observations after removing the orthogonal projections of the observations onto \mathbf{p}_1 . It may be found by solving:

$$\mathbf{p}_2 = \max_{\mathbf{w}_2} \mathbf{w}_2^T \left(\mathbf{Y}_0 - \mathbf{p}_1 \mathbf{p}_1^T \mathbf{Y}_0 \right) \left(\mathbf{Y}_0 - \mathbf{p}_1 \mathbf{p}_1^T \mathbf{Y}_0 \right)^T \mathbf{w}_2 \text{ s.t. } \mathbf{w}_2^T \mathbf{w}_2 = 1. \quad (6.51)$$

⁷That is, $\mathbf{P}^{-1} = \mathbf{P}^T$.

⁸We normalize the eigenvector and disregard its sign.

The solution to Eq. 6.51 is known to be the eigenvector corresponding to the largest eigenvalue under the constraint that it is not collinear with \mathbf{p}_1 . Similarly, the remaining loading vectors are equal to the remaining eigenvectors of $\mathbf{Y}_0 \mathbf{Y}_0^T$ corresponding to descending eigenvalues.

The eigenvalues of $\mathbf{Y}_0 \mathbf{Y}_0^T$, which is a positive semi-definite matrix, are non-negative. They are not necessarily distinct, but since it is a symmetric matrix it has n eigenvectors that are all orthogonal, and it is always diagonalizable. Thus, the matrix \mathbf{P} may be computed by diagonalizing the covariance matrix:

$$\mathbf{Y}_0 \mathbf{Y}_0^T = \mathbf{P} \Lambda \mathbf{P}^{-1} = \mathbf{P} \Lambda \mathbf{P}^T,$$

where $\Lambda = \mathbf{X}_0 \mathbf{X}_0^T$ is a diagonal matrix whose diagonal elements $\{\lambda_i\}_{i=1}^n$ are sorted in descending order.

The transformation back to the observations is $\mathbf{Y} = \mathbf{P}\mathbf{X}$. The fact that the covariance matrix of \mathbf{X} is diagonal means that PCA is a decorrelation transformation and is often used to denoise data.

5.2 Dimensionality Reduction

PCA is often used as a method for dimensionality reduction, the process of reducing the number of variables in a model in order to avoid the curse of dimensionality. PCA gives the first m principal components ($m < n$) by applying the truncated transformation

$$\mathbf{X}_m = \mathbf{P}_m^T \mathbf{Y},$$

where each column of $\mathbf{X}_m \in \mathbb{R}^{m \times N}$ is a vector whose elements are the first m principal components, and \mathbf{P}_m is a matrix whose columns are the first m loading vectors,

$$\mathbf{P}_m = \begin{bmatrix} | & | \\ \mathbf{p}_1 & \cdots & \mathbf{p}_m \\ | & | \end{bmatrix} \in \mathbb{R}^{n \times m}.$$

Intuitively, by keeping only m principal components, we are losing information, and we minimize this loss of information by maximizing their variances.

An important concept in measuring the amount of information lost is the total reconstruction error $\|Y - \hat{Y}\|_F$, where F denotes the Frobenius matrix norm. P_m is also a solution to the minimum total squared reconstruction

$$\min_{W \in \mathbb{R}^{n \times m}} \|Y_0 - WW^T Y_0\|_F^2 \text{ s.t. } W^T W = I_{m \times m}. \quad (6.52)$$

The m leading loading vectors form an orthonormal basis which spans the m dimensional subspace onto which the projections of the demeaned observations have the minimum squared difference from the original demeaned observations.

In other words, P_m compresses each demeaned vector of length n into a vector of length m (where $m \leq n$) in such a way that minimizes the sum of total squared reconstruction errors.

The minimizer of Eq. 6.52 is not unique: $W = P_m Q$ is also a solution, where $Q \in \mathbb{R}^{m \times m}$ is any orthogonal matrix, $Q^T = Q^{-1}$. Multiplying P_m from the right by Q transforms the first m loading vectors into a different orthonormal basis for the same subspace.

6 Summary

This chapter has reviewed foundational material in time series analysis and econometrics. Such material is not intended to substitute more comprehensive and formal treatment of methodology, but rather provide enough background for Chap. 8 where we shall develop neural networks analogues. We have covered the following objectives:

- Explain and analyze linear autoregressive models;
- Understand the classical approaches to identifying, fitting, and diagnosing autoregressive models;
- Apply simple heteroscedastic regression techniques to time series data;
- Understand how exponential smoothing can be used to predict and filter time series; and
- Project multivariate time series data onto lower dimensional spaces with principal component analysis.

It is worth noting that in industrial applications the need to forecast more than a few steps ahead often arises. For example, in algorithmic trading and electronic market making, one needs to forecast far enough into the future, so as to make the forecast economically realizable either through passive trading (skewing of the price) or through aggressive placement of trading orders. This economic realization of the trading signals takes time, whose actual duration is dependent on the frequency of trading.

We should also note that in practice linear regressions predicting the difference between a future and current price, taking as inputs various moving averages, are often used in preference to parametric models, such as GARCH. These linear regressions are often cumbersome, taking as inputs hundreds or thousands of variables.

7 Exercises

Exercise 6.1

Calculate the mean, variance, and autocorrelation function (acf) of the following zero-mean AR(1) process:

$$y_t = \phi_1 y_{t-1} + \epsilon_t,$$

where $\phi_1 = 0.5$. Determine whether the process is stationary by computing the root of the characteristic equation $\Phi(z) = 0$.

Exercise 6.2

You have estimated the following ARMA(1,1) model for some time series data

$$y_t = 0.036 + 0.69 y_{t-1} + 0.42 u_{t-1} + u_t,$$

where you are given the data at time $t - 1$, $y_{t-1} = 3.4$ and $\hat{u}_{t-1} = -1.3$. Obtain the forecasts for the series y for times $t, t + 1, t + 2$ using the estimated ARMA model.

If the actual values for the series are $-0.032, 0.961, 0.203$ for $t, t + 1, t + 2$, calculate the out-of-sample Mean Squared Error (MSE) and Mean Absolute Error (MAE).

Exercise 6.3

Derive the mean, variance, and autocorrelation function (ACF) of a zero mean MA(1) process.

Exercise 6.4

Consider the following log-GARCH(1,1) model with a constant for the mean equation

$$\begin{aligned} y_t &= \mu + u_t, \quad u_t \sim N(0, \sigma_t^2) \\ \ln(\sigma_t^2) &= \alpha_0 + \alpha_1 u_{t-1}^2 + \beta_1 \ln \sigma_{t-1}^2 \end{aligned}$$

- What are the advantages of a log-GARCH model over a standard GARCH model?

- Estimate the unconditional variance of y_t for the values $\alpha_0 = 0.01, \alpha_1 = 0.1, \beta_1 = 0.3$.
- Derive an algebraic expression relating the conditional variance with the unconditional variance.
- Calculate the half-life of the model and sketch the forecasted volatility.

Exercise 6.5

Consider the simple moving average (SMA)

$$S_t = \frac{X_t + X_{t-1} + X_{t+2} + \dots + X_{t-N+1}}{N},$$

and the exponential moving average (EMA), given by $E_1 = X_1$ and, for $t \geq 2$,

$$E_t = \alpha X_t + (1 - \alpha) E_{t-1},$$

where N is the time horizon of the SMA and the coefficient α represents the degree of weighting decrease of the EMA, a constant smoothing factor between 0 and 1. A higher α discounts older observations faster.

- a. Suppose that, when computing the EMA, we stop after k terms, instead of going after the initial value. What fraction of the total weight is obtained?
- b. Suppose that we require 99.9% of the weight. What k do we require?
- c. Show that, by picking $\alpha = 2/(N+1)$, one achieves the same center of mass in the EMA as in the SMA with the time horizon N .
- d. Suppose that we have set $\alpha = 2/(N+1)$. Show that the first N points in an EMA represent about 87.48% of the total weight.

Exercise 6.6

Suppose that, for the sequence of random variables $\{y_t\}_{t=0}^{\infty}$ the following model holds:

$$y_t = \mu + \phi y_{t-1} + \epsilon_t, \quad |\phi| \leq 1, \quad \epsilon_t \sim \text{i.i.d.}(0, \sigma^2).$$

Derive the conditional expectation $\mathbb{E}[y_t | y_0]$ and the conditional variance $\text{Var}[y_t | y_0]$.

Appendix

Hypothesis Tests

Table 6.3 A short summary of some of the most useful diagnostic tests for time series modeling in finance

Name	Description
Chi-squared test	Used to determine whether the confusion matrix of a classifier is statistically significant, or merely white noise
t-test	Used to determine whether the output of two separate regression models are statistically different on i.i.d. data
Mariano-Diebold test	Used to determine whether the output of two separate time series models are statistically different
ARCH test	The ARCH Engle's test is constructed based on the property that if the residuals are heteroscedastic, the squared residuals are autocorrelated. The Ljung–Box test is then applied to the squared residuals
Portmanteau test	A general test for whether the error in a time series model is auto-correlated Example tests include the Box-Ljung and the Box-Pierce test

Python Notebooks

Please see the code folder of Chap. 6 for e.g., implementations of ARIMA models applied to time series prediction. An example, applying PCA to decompose stock prices is also provided in this folder. Further details of these notebooks are included in the README .md file for Chap. 6.

Reference

Tsay, R. S. (2010). *Analysis of financial time series* (3rd ed.). Wiley.

Chapter 7

Probabilistic Sequence Modeling



This chapter presents a powerful class of probabilistic models for financial data. Many of these models overcome some of the severe stationarity limitations of the frequentist models in the previous chapters. The fitting procedure demonstrated is also different—the use of Kalman filtering algorithms for state-space models rather than maximum likelihood estimation or Bayesian inference. Simple examples of hidden Markov models and particle filters in finance and various algorithms are presented.

1 Introduction

So far we have seen how sequences can be modeled using autoregressive processes, moving averages, GARCH, and similar methods. There exists another school of thought, which gave rise to hidden Markov models, Baum–Welch and Viterbi algorithms, Kalman and particle filters.

In this school of thought, one assumes the existence of a certain latent process (say X), which evolves over time (so we may write X_t). This unobservable, latent process drives another, observable process (say Y_t), which we may observe either at all times or at some subset of times.

The evolution of the latent process X_t , as well as the dependence of the observable process Y_t on X_t , may be driven by random factors. We therefore talk about a *stochastic* or *probabilistic* model. We also refer to such a model as a state-space model. The state-space model consists in a description of the evolution of the latent state over time and the dependence of the observables on the latent state.

We have already seen probabilistic methods presented in Chaps. 2 and 3. These methods primarily assume that the data is i.i.d. On the other hand, the time series methods presented in the previous chapter are designed for time series data but are not probabilistic. This chapter shall build on these earlier chapters by considering a

powerful class of models for financial data. Many of these models overcome some of the severe stationarity limitations of the frequentist models in the previous chapters. The fitting procedure is also different—we will see the use of Kalman filtering algorithms for state-space models rather than maximum likelihood estimation or Bayesian inference.

Chapter Objectives

By the end of this chapter, the reader should expect to accomplish the following:

- Formulate hidden Markov models (HMMs) for probabilistic modeling over hidden states;
 - Gain familiarity with the Baum–Welch algorithmic for fitting HMMs to time series data;
 - Use the Viterbi algorithm to find the most likely path;
 - Gain familiarity with state-space models and the application of Kalman filters to fit them; and
 - Apply particle filters to financial time series.
-

2 Hidden Markov Modeling

A hidden Markov model (HMM) is a probabilistic model representing probability distributions over *sequences of observations*. HMMs are the simplest “dynamic” Bayesian network¹ and have proven a powerful model in many applied fields including finance. So far in this book, we have largely considered only i.i.d. observations.² Of course, financial modeling is often seated in a Markovian setting where observations depend on, and only on, the previous observation.

We shall briefly review HMMs in passing as they encapsulate important ideas in probabilistic modeling. In particular, they provide intuition for understanding hidden variables and switching. In the next chapter we shall see the examples of switching in dynamic recurrent neural networks, such as GRUs and LSTMs, which use gating. However, this gating is an implicit modeling step and cannot be controlled explicitly as may be needed for regime switching in finance.

Let us assume at time t that the discrete state, s_t , is hidden from the observer. Furthermore, we shall assume that the hidden state is a Markov process. Note this setup differs from mixture models which treat the hidden variable as i.i.d. The time t observation, y_t , is assumed to be independent of the state at all other times. By the Markovian property, the joint distribution of the sequence of states, $\mathbf{s} := \{s_t\}_{t=1}^T$,

¹Dynamic Bayesian networks models are a graphical model used to model dynamic processes through hidden state evolution.

²With the exception of heteroscedastic modeling in Chap. 6.

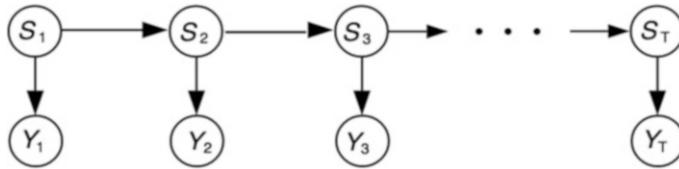


Fig. 7.1 This figure shows the probabilistic graph representing the conditional dependence relations between the observed and the hidden variables in the HMM

and sequence of observations, $\mathbf{y} = \{y_t\}_{t=1}^T$ is given by the product of transition probability densities $p(s_t | s_{t-1})$ and emission probability densities, $p(y_t | s_t)$:

$$p(\mathbf{s}, \mathbf{y}) = p(s_1)p(y_1 | s_1) \prod_{t=2}^T p(s_t | s_{t-1})p(y_t | s_t). \quad (7.1)$$

Figure 7.1 shows the Bayesian network representing the conditional dependence relations between the observed and the hidden variables in the HMM. The conditional dependence relationships define the edges of a graph between parent nodes, Y_t , and child nodes S_t .

Example 7.1 Bull or Bear Market?

Suppose that the market is either in a Bear or Bull market regime represented by $s = 0$ or $s = 1$, respectively. Such states or regimes are assumed to be hidden. Over each period, the market is observed to go up or down and represented by $y = -1$ or $y = 1$. Assume that the emission probability matrix—the conditional dependency matrix between observed and hidden variables—is independent of time and given by

$$\mathbb{P}(y_t = y | s_t = s) = \begin{bmatrix} y/s & 0 & 1 \\ -1 & 0.8 & 0.2 \\ 1 & 0.2 & 0.8 \end{bmatrix}, \quad (7.2)$$

and the transition probability density matrix for the Markov process $\{S_t\}$ is given by

$$A = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}, [A]_{ij} := \mathbb{P}(S_t = s_i | S_{t-1} = s_j). \quad (7.3)$$

Given the observed sequence $\{-1, 1, 1\}$ (i.e., $T = 3$), we can compute the probability of a realization of the hidden state sequence $\{1, 0, 0\}$ using Eq. 7.1. Assuming that $\mathbb{P}(s_1 = 0) = \mathbb{P}(s_1 = 1) = \frac{1}{2}$, the computation is

(continued)

Example 7.1 (continued)

$$\begin{aligned}\mathbb{P}(\mathbf{s}, \mathbf{y}) &= \mathbb{P}(s_1 = 1)\mathbb{P}(y_1 = -1 \mid s_1 = 1)\mathbb{P}(s_2 = 0 \mid s_1 = 1)\mathbb{P}(y_2 = 1 \mid s_2 = 0) \\ &\quad \mathbb{P}(s_3 = 0 \mid s_2 = 0)\mathbb{P}(y_3 = 1 \mid s_3 = 0), \\ &= 0.5 \cdot 0.2 \cdot 0.1 \cdot 0.2 \cdot 0.9 \cdot 0.2 = 0.00036.\end{aligned}$$

We first introduce the so-called *forward* and *backward* quantities, respectively, defined for all states $s_t \in \{1, \dots, K\}$ and over all times

$$F_t(s) := \mathbb{P}(s_t = s, \mathbf{y}_{1:t}) \quad \text{and} \quad B_t(s) := p(\mathbf{y}_{t+1:T} \mid s_t = s) \quad (7.4)$$

with the convention that $B_T(s) = 1$. For all $t \in \{1, \dots, T\}$ and for all $r, s \in \{1, \dots, K\}$ we have

$$\mathbb{P}(s_t = s, \mathbf{y}) = F_t(s)B_t(s), \quad (7.5)$$

and combining the forward and backward quantities gives

$$\mathbb{P}(s_{t-1} = r, s_t = s, \mathbf{y}) = F_{t-1}(r)\mathbb{P}(s_t = s \mid s_{t-1} = r)p(y_t \mid s_t = s)B_t(s). \quad (7.6)$$

The forward–backward algorithm, also known as the *Baum–Welch* algorithm, is an *unsupervised* learning algorithm for fitting HMMs which belongs to the class of EM algorithms.

2.1 The Viterbi Algorithm

In addition to finding the probability of the realization of a particular hidden state sequence, we may also seek the most likely sequence realization. This sequence can be estimated using the Viterbi algorithm.

Suppose once again that we observe a sequence of T *observations*,

$$\mathbf{y} = \{y_1, \dots, y_T\}.$$

However, for each $1 \leq t \leq T$, $y_t \in O$, where $O = \{o_1, o_2, \dots, o_N\}$, $N \in \mathbb{N}$, is now in some *observation space*.

We suppose that, for each $1 \leq t \leq T$, the observation y_t is driven by a (*hidden*) *state* $s_t \in \mathcal{S}$, where $\mathcal{S} := \{s_1, \dots, s_K\}$, $K \in \mathbb{N}$, is some *state space*. For example, y_t might be the credit rating of a corporate bond and s_t might indicate some latent variable such as the overall health of the relevant industry sector.

Given \mathbf{y} , what is the most likely sequence of hidden states,

$$\mathbf{x} = \{x_1, x_2, \dots, x_T\}?$$

To answer this question, we need to introduce a few more constructs. First, the set of *initial probabilities* must be given:

$$\boldsymbol{\pi} = \{\pi_1, \dots, \pi_K\},$$

so that π_i is the probability that $s_1 = f_i$, $1 \leq i \leq K$.

We also need to specify the *transition matrix* $A \in \mathbb{R}^{K \times K}$, such that the element A_{ij} , $1 \leq i, j \leq K$ is the transition probability of transitioning from state f_i to state f_j .

Finally, we need the *emission matrix* $B \in \mathbb{R}^{K \times N}$, such that the element B_{ij} , $1 \leq i \leq K$, $1 \leq j \leq N$ is the probability of observing o_j from state f_i .

Let us now consider a simple example to fix ideas.

Example 7.2 The Crooked Dealer

A dealer has two coins, a fair coin, with $\mathbb{P}(\text{Heads}) = \frac{1}{2}$, and a loaded coin, with $\mathbb{P}(\text{Heads}) = \frac{4}{5}$. The dealer starts with the fair coin with probability $\frac{3}{5}$. The dealer then tosses the coin several times. After each toss, there is a $\frac{2}{5}$ probability of a switch to the other coin. The observed sequence is Heads, Tails, Heads, Tails, Heads, Heads, Tails, Heads.

In this case, the state space and observation space are, respectively,

$$\mathcal{S} = \{f_1 = \text{Fair}, f_2 = \text{Loaded}\}, \quad \mathcal{O} = \{o_1 = \text{Heads}, o_2 = \text{Tails}\},$$

with initial probabilities

$$\boldsymbol{\pi} = \{\pi_1 = 0.6, \pi_2 = 0.4\},$$

transition probabilities

$$A = \begin{pmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{pmatrix},$$

and the emission matrix is

$$B = \begin{pmatrix} 0.5 & 0.5 \\ 0.8 & 0.2 \end{pmatrix}.$$

Given the sequence of observations

$$\mathbf{y} = (\text{Heads}, \text{Tails}, \text{Heads}, \text{Tails}, \text{Heads}, \text{Heads}, \text{Tails}, \text{Heads}),$$

we would like to find the most likely sequence of hidden states, $\mathbf{s} = \{s_1, \dots, s_T\}$, i.e., determine which of the two coins generated which of the coin tosses.

One way to answer this question is by applying the *Viterbi algorithm* as detailed in the notebook `Viterbi.ipynb`. We note that the most likely state sequence \mathbf{s} , which produces the observation sequence $\mathbf{y} = \{y_1, \dots, y_T\}$, satisfies the recurrence relations

$$\begin{aligned} V_{1,k} &= \mathbb{P}(y_1 | s_1 = j_k) \cdot \pi_k, \\ V_{t,k} &= \max_{1 \leq i \leq K} (\mathbb{P}(y_t | s_t = j_k) \cdot A_{ik} \cdot V_{t-1,i}), \end{aligned}$$

where $V_{t,k}$ is the probability of the most probable state sequence $\{s_1, \dots, s_t\}$ such that $s_t = j_k$,

$$V_{t,k} = \mathbb{P}(s_1, \dots, s_t, y_1, \dots, y_t | s_t = j_k).$$

The actual *Viterbi path* can be obtained by, at each step, keeping track of which state index i was used in the second equation. Let $\xi(k, t)$ be the function that returns the value of i that was used to compute $V_{t,k}$ if $t > 1$, or k if $t = 1$. Then

$$\begin{aligned} s_T &= \arg \max_{1 \leq i \leq K} (V_{T,k}), \\ s_{t-1} &= \xi(s_t, t). \end{aligned}$$

We leave the application of the Viterbi algorithm to our example as an exercise for the reader.

Note that the Viterbi algorithm determines the most likely complete sequence of hidden states given the sequence of observations and the model specification, including the *known* transition and emission matrices. If these matrices are known, there is no reason to use the Baum–Welch algorithm. If they are unknown, then the Baum–Welch algorithm must be used.

2.1.1 Filtering and Smoothing with HMMs

Financial data is typically noisy and we need techniques which can extract the signal from the noise. There are many techniques for reducing the noise. Filtering is a general term for extracting information from a noisy signal. Smoothing is a particular kind of filtering in which low-frequency components are passed and high-frequency components are attenuated. Filtering and smoothing produce distributions of states at each time step. Whereas maximum likelihood estimation chooses the state with the highest probability at the “best” estimate at each time step, this may not lead to the best path in HMMs. We have seen that the Baum–Welch algorithm can be deployed to find the optimal state trajectory, not just the optimal sequence of “best” states.

2.2 State-Space Models

HMMs belong in the same class as linear Gaussian state-space models. These are known as “Kalman filters” which are *continuous* latent state analogues of HMMs. Note that we have already seen examples of continuous state-space models, which are not necessarily Gaussian, in our exposition on RNNs in Chap. 8.

The state transition probability $p(s_t | s_{t-1})$ can be decomposed into deterministic and noise:

$$s_t = F_t(s_{t-1}) + \epsilon_t, \quad (7.7)$$

for some deterministic function and ϵ_t is zero-mean i.i.d. noise. Similarly, the emission probability $p(y_t | s_t)$ can be decomposed as:

$$y_t = G_t(s_t) + \xi_t, \quad (7.8)$$

with zero-mean i.i.d. observation noise. If the functions F_t and G_t are linear and time independent, then we have

$$s_t = As_{t-1} + \epsilon_t, \quad (7.9)$$

$$y_t = Cs_t + \xi_t, \quad (7.10)$$

where A is the state transition matrix and C is the observation matrix. For completeness, we contrast the Kalman filter with a univariate RNN, as described in Chap. 8. When the observations are predictors, x_t , and the hidden variables are s_t we have

$$s_t = F(s_{t-1}, y_t) := \sigma(As_{t-1} + By_t), \quad (7.11)$$

$$y_t = Cs_t + \xi_t, \quad (7.12)$$

where we have ignored the bias terms for simplicity. Hence, the RNN state equation differs from the Kalman filter in that (i) it is a non-linear function of both the previous state and the observation; and (ii) it is noise-free.

3 Particle Filtering

A Kalman filter maintains its state as moments of the multivariate Gaussian distribution, $\mathcal{N}(\mathbf{m}, \mathbf{P})$. This approach is appropriate when the state is Gaussian, or when the true distribution can be closely approximated by the Gaussian. What if the distribution is, for example, bimodal?

Arguably the simplest way to approximate more or less any distribution, including a bimodal distribution, is by data points sampled from that distribution. We refer to those data points as “particles.”

The more particles we have, the more closely we can approximate the target distribution. The approximate, empirical distribution is then given by the histogram. Note that the particles need not be univariate, as in our example. They may be multivariate if we are approximating a multivariate distribution. Also, in our example the particles all have the same weight. More generally, we may consider weighted particles, whose weights are unequal.

This setup gives rise to the family of algorithms known as *particle filtering* algorithms (Gordon et al. 1993; Kitagawa 1993). One of the most common of them is the *Sequential Importance Resampling (SIR)* algorithm:

3.1 Sequential Importance Resampling (SIR)

- Initialization step:* At time $t = 0$, draw M i.i.d. samples from the initial distribution τ_0 . Also, initialize M normalized (to 1) weights to an identical value $\frac{1}{M}$. For $i = 1, \dots, M$, the samples will be denoted $\hat{\mathbf{x}}_{0|0}^{(i)}$ and the normalized weights $\lambda_0^{(i)}$.
- Recursive step:* At time $t = 1, \dots, T$, let $(\hat{\mathbf{x}}_{t-1|t-1}^{(i)})_{i=1,\dots,M}$ be the particles generated at time $t - 1$.
 - *Importance sampling:* For $i = 1, \dots, M$, sample $\hat{\mathbf{x}}_{t|t-1}^{(i)}$ from the Markov transition kernel $\tau_t(\cdot | \hat{\mathbf{x}}_{t-1|t-1}^{(i)})$. For $i = 1, \dots, M$, use the observation density to compute the non-normalized weights

$$\omega_t^{(i)} := \lambda_{t-1}^{(i)} \cdot p(\mathbf{y}_t | \hat{\mathbf{x}}_{t|t-1}^{(i)})$$

and the values of the normalized weights before resampling (“br”)

$$\text{br}\lambda_t^{(i)} := \frac{\omega_t^{(i)}}{\sum_{k=1}^M \omega_t^{(k)}}.$$

- *Resampling (or selection):* For $i = 1, \dots, M$, use an appropriate resampling algorithm (such as *multinomial resampling*—see below) to sample $\mathbf{x}_{t|t}^{(i)}$ from the mixture

$$\sum_{k=1}^M \text{br}\lambda_t^{(k)} \delta(\mathbf{x}_t - \mathbf{x}_{t|t-1}^{(k)}),$$

where $\delta(\cdot)$ denotes the Dirac delta generalized function, and set the normalized weights after resampling, $\lambda_t^{(i)}$, appropriately (for most common resampling algorithms this means $\lambda_t^{(i)} := \frac{1}{M}$).

Informally, SIR shares some of the characteristics of genetic algorithms; based on the likelihoods $p(y_t | \hat{x}_{t|t-1}^{(i)})$, we increase the weights of the more “successful” particles, allowing them to “thrive” at the resampling step.

The resampling step was introduced to avoid the degeneration of the particles, with all the weight concentrating on a single point. The most common resampling scheme is the so-called *multinomial resampling* which we now review.

3.2 Multinomial Resampling

Notice, from above, that we are using with the *normalized* weights computed before resampling, ${}^{\text{br}}\lambda_t^{(1)}, \dots, {}^{\text{br}}\lambda_t^{(M)}$:

- a. For $i = 1, \dots, M$, compute the cumulative sums

$${}^{\text{br}}\Lambda_t^{(i)} = \sum_{k=1}^i {}^{\text{br}}\lambda_t^{(k)},$$

so that, by construction, ${}^{\text{br}}\Lambda_t^{(M)} = 1$.

- b. Generate M random samples from $\mathcal{U}(0, 1), u_1, u_2, \dots, u_M$.
- c. For each $i = 1, \dots, M$, choose the particle $\hat{x}_{t|t}^{(i)} = \hat{x}_{t|t-1}^{(j)}$ with $j \in \{1, 2, \dots, M-1\}$ such that $u_i \in [{}^{\text{br}}\Lambda_t^{(j)}, {}^{\text{br}}\Lambda_t^{(j+1)}]$.

Thus we end up with M new particles (*children*), $\mathbf{x}_{t|t}^{(1)}, \dots, \mathbf{x}_{t|t}^{(M)}$ sampled from the existing set $\mathbf{x}_{t|t-1}^{(1)}, \dots, \mathbf{x}_{t|t-1}^{(M)}$, so that some of the existing particles may disappear, while others may appear multiple times. For each $i = 1, \dots, M$ the number of times $\mathbf{x}_{t|t-1}^{(i)}$ appears in the resampled set of particles is known as the particle’s *replication factor*, $N_t^{(i)}$.

We set the normalized weights after resampling: $\lambda_t^{(i)} := \frac{1}{M}$. We could view this algorithm as the sampling of the replication factors $N_t^{(1)}, \dots, N_t^{(M)}$ from the multinomial distribution with probabilities ${}^{\text{br}}\lambda_t^{(1)}, \dots, {}^{\text{br}}\lambda_t^{(M)}$, respectively. Hence the name of the method. ■

3.3 Application: Stochastic Volatility Models

Stochastic Volatility (SV) models have been studied extensively in the literature, often as applications of *particle filtering* and *Markov chain Monte Carlo (MCMC)*. Their broad appeal in finance is their ability to capture the “leverage effect”—the observed tendency of an asset’s volatility to be negatively correlated with the asset’s returns (Black 1976).

In particular, Pitt, Malik, and Doucet apply the particle filter to the *stochastic volatility with leverage and jumps (SVLJ)* (Malik and Pitt 2009, 2011a,b; Pitt et al. 2014). The model has the general form of Taylor (1982) with two modifications. For $t \in \mathbb{N} \cup \{0\}$, let y_t denote the log-return on an asset and x_t denote the log-variance of that return. Then

$$y_t = \epsilon_t e^{x_t/2} + J_t \varpi_t, \quad (7.13)$$

$$x_{t+1} = \mu(1 - \phi) + \phi x_t + \sigma_v \eta_t, \quad (7.14)$$

where μ is the mean log-variance, ϕ is the persistence parameter, σ_v is the volatility of log-variance.

The first modification to Taylor’s model is the introduction of correlation between ϵ_t and η_t :

$$\begin{pmatrix} \epsilon_t \\ \eta_t \end{pmatrix} \sim \mathcal{N}(0, \Sigma), \quad \Sigma = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}.$$

The correlation ρ is the *leverage* parameter. In general, $\rho < 0$, due to the leverage effect.

The second change is the introduction of jumps. $J_t \in \{0, 1\}$ is a Bernoulli counter with intensity p (thus p is the jump intensity parameter), $\varpi_t \sim \mathcal{N}(0, \sigma_J^2)$ determines the jump size (thus σ_J is the jump volatility parameter).

We obtain a *stochastic volatility with leverage (SVL)*, but no jumps, if we delete the $J_t \varpi_t$ term or, equivalently, set p to zero. Taylor’s original model is a special case of SVLJ with $p = 0$, $\rho = 0$.

This, then, leads to the following adaptation of SIR, developed by Doucet, Malik, and Pitt, for this special case with nonadditive, correlated noises. The initial distribution of x_0 is taken to be $\mathcal{N}(0, \sigma_v^2 / (1 - \phi^2))$.

- a. *Initialization step:* At time $t = 0$, draw M i.i.d. particles from the initial distribution $\mathcal{N}(0, \sigma_v^2 / (1 - \phi^2))$. Also, initialize M normalized (to 1) weights to an identical value of $\frac{1}{M}$. For $i = 1, 2, \dots, M$, the samples will be denoted $\hat{x}_0^{(i)}$ and the normalized weights $\lambda_0^{(i)}$.
- b. *Recursive step:* At time $t \in \mathbb{N}$, let $(\hat{x}_{t-1}^{(i)})_{i=1,\dots,M}$ be the particles generated at time $t - 1$.

i *Importance sampling:*

- First,
 - For $i = 1, \dots, M$, sample $\hat{\epsilon}_{t-1}^{(i)}$ from $p(\epsilon_{t-1} | x_{t-1} = \hat{x}_{t-1|t-1}^{(i)}, y_{t-1})$. (If no y_{t-1} is available, as at $t = 1$, sample from $p(\epsilon_{t-1} | x_{t-1} = \hat{x}_{t-1|t-1}^{(i)})$).
 - For $i = 1, \dots, M$, sample $\hat{x}_{t|t-1}^{(i)}$ from $p(x_t | x_{t-1} = \hat{x}_{t-1|t-1}^{(i)}, y_{t-1}, \hat{\epsilon}_{t-1}^{(i)})$.
- For $i = 1, \dots, M$, compute the non-normalized weights:

$$\omega_t^{(i)} := \lambda_{t-1}^{(i)} \cdot p_{\gamma_t}(y_t | \hat{x}_{t|t-1}^{(i)}), \quad (7.15)$$

using the observation density

$$p(y_t | \hat{x}_{t|t-1}^{(i)}, p, \sigma_J^2) = (1-p) \left[\left(2\pi e^{\hat{x}_{t|t-1}^{(i)}} \right)^{-1/2} \exp \left(-y_t^2 / (2e^{\hat{x}_{t|t-1}^{(i)}}) \right) \right] + \\ p \left[\left(2\pi (e^{\hat{x}_{t|t-1}^{(i)}} + \sigma_J^2) \right)^{-1/2} \exp \left(-y_t^2 / (2e^{\hat{x}_{t|t-1}^{(i)}} + 2\sigma_J^2) \right) \right],$$

and the values of the normalized weights before resampling ('br'):

$$\text{br}\lambda_t^{(i)} := \frac{\omega_t^{(i)}}{\sum_{k=1}^M \omega_t^{(k)}}.$$

ii *Resampling* (or *selection*): For $i = 1, \dots, M$, use an appropriate resampling algorithm (such as multinomial resampling) sample $\hat{x}_{t|t}^{(i)}$ from the mixture

$$\sum_{k=1}^M \text{br}\lambda_t^{(k)} \delta(x_t - \hat{x}_{t|t-1}^{(k)}),$$

where $\delta(\cdot)$ denotes the Dirac delta generalized function, and set the normalized weights after resampling, $\lambda_t^{(i)}$, according to the resampling algorithm.

4 Point Calibration of Stochastic Filters

We have seen in the example of the stochastic volatility model with leverage and jumps (SVLJ) that the state-space model may be parameterized by a parameter vector, $\theta \in \mathbb{R}^{d_\theta}$, $d_\theta \in \mathbb{N}$. In that particular case,

$$\boldsymbol{\theta} = \begin{pmatrix} \mu \\ \phi \\ \sigma_\eta^2 \\ \rho \\ \sigma_J^2 \\ p \end{pmatrix}.$$

We may not know the true value of this parameter. How do we estimate it? In other words, how do we *calibrate* the model, given a time series of either historical or generated observations, $\mathbf{y}_1, \dots, \mathbf{y}_T, T \in \mathbb{N}$.

The frequentist approach relies on the (joint) probability density function of the observations, which depends on the parameters, $p(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T | \boldsymbol{\theta})$. We can regard this as a function of $\boldsymbol{\theta}$ with $\mathbf{y}_1, \dots, \mathbf{y}_T$ fixed, $p(\mathbf{y}_1, \dots, \mathbf{y}_T | \boldsymbol{\theta}) =: \mathcal{L}(\boldsymbol{\theta})$ —the *likelihood function*.

This function is sometimes referred to as *marginal likelihood*, since the hidden states, $\mathbf{x}_1, \dots, \mathbf{x}_T$, are marginalized out. We seek a *maximum likelihood estimator (MLE)*, $\hat{\boldsymbol{\theta}}_{ML}$, the value of $\boldsymbol{\theta}$ that maximizes the likelihood function.

Each evaluation of the objective function, $\mathcal{L}(\boldsymbol{\theta})$, constitutes a run of the stochastic filter over the observations $\mathbf{y}_1, \dots, \mathbf{y}_T$. By the chain rule (i), and since we use a Markov chain (ii),

$$p(\mathbf{y}_1, \dots, \mathbf{y}_T) \stackrel{(i)}{=} \prod_{t=1}^T p(\mathbf{y}_t | \mathbf{y}_0, \dots, \mathbf{y}_{t-1}) \stackrel{(ii)}{=} \prod_{t=1}^T \int p(\mathbf{y}_t | \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{y}_0, \dots, \mathbf{y}_{t-1}) d\mathbf{x}_t.$$

Note that, for ease of notation, we have omitted the dependence of all the probability densities on $\boldsymbol{\theta}$, e.g., instead of writing $p(\mathbf{y}_1, \dots, \mathbf{y}_T; \boldsymbol{\theta})$.

For the particle filter, we can estimate the log-likelihood function from the non-normalized weights:

$$p(\mathbf{y}_1, \dots, \mathbf{y}_T) = \prod_{t=1}^T \int p(\mathbf{y}_t | \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{y}_0, \dots, \mathbf{y}_{t-1}) d\mathbf{x}_t \approx \prod_{t=1}^T \left(\frac{1}{M} \sum_{k=1}^M \omega_t^{(k)} \right),$$

whence

$$\ln(\mathcal{L}(\boldsymbol{\theta})) = \ln \left\{ \prod_{t=1}^T \left(\frac{1}{M} \sum_{k=1}^M \omega_t^{(k)} \right) \right\} = \sum_{t=1}^T \ln \left(\frac{1}{M} \sum_{k=1}^M \omega_t^{(k)} \right). \quad (7.16)$$

This was first proposed by Kitagawa (1993, 1996) for the purposes of approximating $\hat{\boldsymbol{\theta}}_{ML}$.

In most practical applications one needs to resort to numerical methods, perhaps quasi-Newton methods, such as *Broyden–Fletcher–Goldfarb–Shanno (BFGS)* (Gill et al. 1982), to find $\hat{\boldsymbol{\theta}}_{ML}$.

Pitt et al. (2014) point out the practical difficulties which result when using the above as an objective function in an optimizer. In the resampling (or selection) step of the particle filter, we are sampling from a discontinuous empirical distribution function. Therefore, $\ln(\mathcal{L}(\theta))$ will not be continuous as a function of θ . To remedy this, they rely on an alternative, continuous, resampling procedure. A quasi-Newton method is then used to find $\hat{\theta}_{ML}$ for the parameters $\theta = (\mu, \phi, \sigma_v^2, \rho, p, \sigma_j^2)^\top$ of the SVLJ model.

We note in passing that Kalman filters can also be calibrated using a similar maximum likelihood approach.

5 Bayesian Calibration of Stochastic Filters

Let us briefly discuss how filtering methods relate to *Markov chain Monte Carlo methods (MCMC)*—a vast subject in its own right; therefore, our discussion will be cursory at best. The technique takes its origin from Metropolis et al. (1953).

Following Kim et al. (1998) and Meyer and Yu (2000); Yu (2005), we demonstrate how MCMC techniques can be used to estimate the parameters of the SVL model. They calibrate the parameters to the time series of observations of daily mean-adjusted log-returns, y_1, \dots, y_T to obtain the joint prior density

$$p(\theta, x_0, \dots, x_T) = p(\theta)p(x_0 | \theta) \prod_{t=1}^T p(x_t | x_{t-1}, \theta)$$

by successive conditioning. Here $\theta := (\mu, \phi, \sigma_v^2, \rho)^\top$ is, as before, the vector of the model parameters. We assume prior independence of the parameters and choose the same priors (as in Kim et al. (1998)) for μ , ϕ , and σ_v^2 , and a uniform prior for ρ . The observation model and the conditional independence assumption give the likelihood

$$p(y_1, \dots, y_T | \theta, x_0, \dots, x_T) = \prod_{t=1}^T p(y_t | x_t),$$

and the joint posterior distribution of the *unobservables* (the parameters θ and the hidden states x_0, \dots, x_T ; in the Bayesian perspective these are treated identically and estimated in a similar manner) follows from Bayes' theorem; for the SVL model, this posterior satisfies

$$\begin{aligned} p(\theta, x_0, \dots, x_T | y_1, \dots, y_T) &\propto p(\mu)p(\phi)p(\sigma_v^2)p(\rho) \\ &\quad \prod_{t=1}^T p(x_{t+1} | x_t, \mu, \phi, \sigma_v^2) \prod_{t=1}^T p(y_t | x_{t+1}, x_t, \mu, \phi, \sigma_v^2, \rho), \end{aligned}$$

where $p(\mu)$, $p(\phi)$, $p(\sigma_v^2)$, $p(\rho)$ are the appropriately chosen priors,

$$x_{t+1} | x_t, \mu, \phi, \sigma_v^2 \sim \mathcal{N}\left(\mu(1 - \phi) + \phi x_t, \sigma_v^2\right),$$

$$y_t | x_{t+1}, x_t, \mu, \phi, \sigma_v^2, \rho \sim \mathcal{N}\left(\frac{\rho}{\sigma_v} e^{x_t/2} (x_{t+1} - \mu(1 - \phi) - \phi x_t), e^{x_t}(1 - \rho^2)\right).$$

Meyer and Yu use the software package **BUGS**³ (Spiegelhalter et al. 1996; Lunn et al. 2000) represent the resulting Bayesian model as a *directed acyclic graph (DAG)*, where the nodes are either constants (denoted by rectangles), stochastic nodes (variables that are given a distribution, denoted by ellipses), or deterministic nodes (logical functions of other nodes); the arrows either indicate stochastic dependence (solid arrows) or logical functions (hollow arrows). This graph helps visualize the conditional (in)dependence assumptions and is used by **BUGS** to construct full univariate conditional posterior distributions for all unobservables. It then uses Markov chain Monte Carlo algorithms to sample from these distributions.

The algorithm based on the original work (Metropolis et al. 1953) is now known as the *Metropolis algorithm*. It has been generalized by Hastings (1930–2016) to obtain the *Metropolis–Hastings algorithm* (Hastings 1970) and further by Green to obtain what is known as the *Metropolis–Hastings–Green algorithm* (Green 1995). A popular algorithm based on a special case of the Metropolis–Hastings algorithm, known as the *Gibbs sampler*, was developed by Geman and Geman (1984) and, independently, Tanner and Wong (1987).⁴ It was further popularized by Gelfand and Smith (1990). Gibbs sampling and related algorithms (Gilks and Wild 1992; Ritter and Tanner 1992) are used by **BUGS** to sample from the univariate conditional posterior distributions for all unobservables. As a result we perform Bayesian estimation—obtain estimates of the *distributions* of the parameters μ , ϕ , σ_v^2 , ρ —rather than frequentist estimation, where a single value of the parameters vector, which maximizes the likelihood, $\hat{\theta}_{ML}$, is produced. Stochastic filtering, sometimes in combination with MCMC, can be used for both frequentist and Bayesian parameter estimation (Chen 2003). Filtering methods that update estimates of the parameters online, while processing observations in real-time, are referred to as *adaptive filtering* (see Sayed (2008); Vega and Rey (2013); Crisan and Miguez (2013); Naesseth et al. (2015) and references therein).

We note that a Gibbs sampler (or variants thereof) is a highly nontrivial piece of software. In addition to the now classical **BUGS**/Win**BUGS** there exist powerful Gibbs samplers accessible via modern libraries, such as Stan, Edward, and PyMC3.

³An acronym for Bayesian inference Using Gibbs Sampling.

⁴Sometimes the Gibbs sampler is referred to as *data augmentation* following this paper.

6 Summary

This chapter extends Chap. 2 by presenting probabilistic methods for time series data. The key modeling assumption is the existence of a certain latent process X_t , which evolves over time. This unobservable, latent process drives another, observable process. Such an approach overcomes limitations of stationarity imposed on the methods in the previous chapter. The reader should verify that they have achieved the primary learning objectives of this chapter:

- Formulate hidden Markov models (HMMs) for probabilistic modeling over hidden states;
- Gain familiarity with the Baum–Welch algorithm for fitting HMMs to time series data;
- Use the Viterbi algorithm to find the most likely path;
- Gain familiarity with state-space models and the application of Kalman filters to fit them; and
- Apply particle filters to financial time series.

7 Exercises

Exercise 7.1: Kalman Filtering of Autoregressive Moving Average ARMA(p, q) Model

The *autoregressive moving average* ARMA(p, q) model can be written as

$$y_t = \phi_1 y_{t-1} + \dots + \phi_p y_{t-p} + \eta_t + \theta_1 \eta_{t-1} + \dots + \theta_q \eta_{t-q},$$

where $\eta_t \sim \mathcal{N}(0, \sigma^2)$ and includes as special cases all AR(p) and MA(q) models. Such models are often fitted to financial time series. Suppose that we would like to filter this time series using a Kalman filter. Write down a suitable process and the observation models.

Exercise 7.2: The Ornstein–Uhlenbeck Process

Consider the one-dimensional *Ornstein–Uhlenbeck (OU)* process, the stationary Gauss–Markov process given by the SDE

$$dX_t = \theta(\mu - X_t) dt + \sigma dW_t,$$

where $X_t \in \mathbb{R}$, $X_0 = x_0$, and $\theta > 0$, μ , and $\sigma > 0$ are constants. Formulate the Kalman process model for this process.

Exercise 7.3: Deriving the Particle Filter for Stochastic Volatility with Leverage and Jumps

We shall regard the log-variance x_t as the hidden states and the log-returns y_t as observations. How can we use the particle filter to estimate x_t on the basis of the observations y_t ?

- Show that, in the absence of jumps,

$$x_t = \mu(1 - \phi) + \phi x_{t-1} + \sigma_v \rho y_{t-1} e^{-x_{t-1}/2} + \sigma_v \sqrt{1 - \rho^2} \xi_{t-1}$$

for some $\xi_t \stackrel{i.i.d.}{\sim} \mathcal{N}(0, 1)$.

- Show that

$$\begin{aligned} p(\epsilon_t | x_t, y_t) &= \delta(\epsilon_t - y_t e^{-x_t/2}) \mathbb{P}[J_t = 0 | x_t, y_t] \\ &\quad + \phi(\epsilon_t; \mu_{\epsilon_t | J_t=1}, \sigma_{\epsilon_t | J_t=1}^2) \mathbb{P}[J_t = 1 | x_t, y_t], \end{aligned}$$

where

$$\mu_{\epsilon_t | J_t=1} = \frac{y_t \exp(x_t/2)}{\exp(x_t) + \sigma_J^2}$$

and

$$\sigma_{\epsilon_t | J_t=1}^2 = \frac{\sigma_J^2}{\exp(x_t) + \sigma_J^2}.$$

- Explain how you could implement random sampling from the probability distribution given by the density $p(\epsilon_t | x_t, y_t)$.
- Write down the probability density $p(x_t | x_{t-1}, y_{t-1}, \epsilon_{t-1})$.
- Explain how you could sample from this distribution.
- Show that the observation density is given by

$$\begin{aligned} p(y_t | \hat{x}_{t|t-1}^{(i)}, p, \sigma_J^2) &= (1-p) \left[\left(2\pi e^{\hat{x}_{t|t-1}^{(i)}} \right)^{-1/2} \exp \left(-y_t^2 / (2e^{\hat{x}_{t|t-1}^{(i)}}) \right) \right] + \\ &p \left[\left(2\pi (e^{\hat{x}_{t|t-1}^{(i)}} + \sigma_J^2) \right)^{-1/2} \exp \left(-y_t^2 / (2e^{\hat{x}_{t|t-1}^{(i)}} + 2\sigma_J^2) \right) \right]. \end{aligned}$$

Exercise 7.4: The Viterbi Algorithm and an Occasionally Dishonest Casino

The dealer has two coins, a fair coin, with $\mathbb{P}(\text{Heads}) = \frac{1}{2}$, and a loaded coin, with $\mathbb{P}(\text{Heads}) = \frac{4}{5}$. The dealer starts with the fair coin with probability $\frac{3}{5}$. The dealer then tosses the coin several times. After each toss, there is a $\frac{2}{5}$ probability of a switch to the other coin. The observed sequence is Heads, Tails, Tails, Heads, Tails, Heads, Heads, Heads, Tails, Heads. Run the Viterbi algorithm to determine which coin the dealer was most likely using for each coin toss.

Appendix

Python Notebooks

The notebooks provided in the accompanying source code repository are designed to gain familiarity with how to implement the Viterbi algorithm and particle filtering for stochastic volatility model calibration. Further details of the notebooks are included in the README.md file.

References

- Black, F. (1976). Studies of stock price volatility changes. In *Proceedings of the Business and Economic Statistics Section*.
- Chen, Z. (2003). Bayesian filtering: From Kalman filters to particle filters, and beyond. *Statistics*, 182(1), 1–69.
- Crisan, D., & Míguez, J. (2013). Nested particle filters for online parameter estimation in discrete-time state-space Markov models. *ArXiv:1308.1883*.
- Gelfand, A. E., & Smith, A. F. M. (1990, June). Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410), 398–409.
- Geman, S. J., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6, 721–741.
- Gilks, W. R., & Wild, P. P. (1992). Adaptive rejection sampling for Gibbs sampling, Vol. 41, pp. 337–348.
- Gill, P. E., Murray, W., & Wright, M. H. (1982). *Practical optimization*. Emerald Group Publishing Limited.
- Gordon, N. J., Salmond, D. J., & Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*.
- Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4), 711–32.
- Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1), 97–109.
- Kim, S., Shephard, N., & Chib, S. (1998, July). Stochastic volatility: Likelihood inference and comparison with ARCH models. *The Review of Economic Studies*, 65(3), 361–393.
- Kitagawa, G. (1993). A Monte Carlo filtering and smoothing method for non-Gaussian nonlinear state space models. In *Proceedings of the 2nd U.S.-Japan Joint Seminar on Statistical Time Series Analysis* (pp. 110–131).
- Kitagawa, G. (1996). Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5(1), 1–25.
- Lunn, D. J., Thomas, A., Best, N. G., & Spiegelhalter, D. (2000). WinBUGS – a Bayesian modelling framework: Concepts, structure and extensibility. *Statistics and Computing*, 10, 325–337.
- Malik, S., & Pitt, M. K. (2009, April). *Modelling stochastic volatility with leverage and jumps: A simulated maximum likelihood approach via particle filtering*. Warwick Economic Research Papers 897, The University of Warwick, Department of Economics, Coventry CV4 7AL.
- Malik, S., & Pitt, M. K. (2011a, February). *Modelling stochastic volatility with leverage and jumps: A simulated maximum likelihood approach via particle filtering*. document de travail 318, Banque de France Eurostème.

- Malik, S., & Pitt, M. K. (2011b). Particle filters for continuous likelihood evaluation and maximisation. *Journal of Econometrics*, 165, 190–209.
- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21.
- Meyer, R., & Yu, J. (2000). BUGS for a Bayesian analysis of stochastic volatility models. *Econometrics Journal*, 3, 198–215.
- Naesseth, C. A., Lindsten, F., & Schön, T. B. (2015). Nested sequential Monte Carlo methods. In *Proceedings of the 32nd International Conference on Machine Learning*.
- Pitt, M. K., Malik, S., & Doucet, A. (2014). Simulated likelihood inference for stochastic volatility models using continuous particle filtering. *Annals of the Institute of Statistical Mathematics*, 66, 527–552.
- Ritter, C., & Tanner, M. A. (1992). Facilitating the Gibbs sampler: The Gibbs stopper and the Griddy-Gibbs sampler. *Journal of the American Statistical Association*, 87(419), 861–868.
- Sayed, A. H. (2008). *Adaptive filters*. Wiley-Interscience.
- Spiegelhalter, D., Thomas, A., Best, N. G., & Gilks, W. R. (1996, August). *BUGS 0.5: Bayesian inference using Gibbs sampling manual (version ii)*. Robinson Way, Cambridge CB2 2SR: MRC Biostatistics Unit, Institute of Public Health.
- Tanner, M. A., & Wong, W. H. (1987, June). The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, 82(398), 528–540.
- Taylor, S. J. (1982). *Time series analysis: theory and practice*. Chapter Financial returns modelled by the product of two stochastic processes, a study of daily sugar prices, pp. 203–226. North-Holland.
- Vega, L. R., & H. Rey (2013). *A rapid introduction to adaptive filtering*. Springer Briefs in Electrical and Computer Engineering. Springer.
- Yu, J. (2005). On leverage in a stochastic volatility model. *Journal of Econometrics*, 127, 165–178.

Chapter 8

Advanced Neural Networks



This chapter presents various neural network models for financial time series analysis, providing examples of how they relate to well-known techniques in financial econometrics. Recurrent neural networks (RNNs) are presented as non-linear time series models and generalize classical linear time series models such as $AR(p)$. They provide a powerful approach for prediction in financial time series and generalize to non-stationary data. This chapter also presents convolution neural networks for filtering time series data and exploiting different scales in the data. Finally, this chapter demonstrates how autoencoders are used to compress information and generalize principal component analysis.

1 Introduction

The universal approximation theorem states that a feedforward network is capable of approximating any function. So why do other types of neural networks exist? One answer to this is efficiency. In this chapter, different architectures shall be explored for their ability to exploit the structure in the data, resulting in fewer weights. Hence the main motivation for different architectures is often parsimony of parameters and therefore less propensity to overfit and reduced training time. We shall see that other architectures can be used, in particular ones that change their behavior over time, without the need to retrain the networks. And we will see how neural networks can be used to compress data, analogously to principal component analysis.

There are other neural network architectures which are used in financial applications but are too esoteric to list here. However, we shall focus on three other classes of neural networks which have proven to be useful in the finance industry. The first two are supervised learning techniques and the latter is an unsupervised learning technique.

Recurrent neural networks (RNNs) are non-linear time series models and generalize classical linear time series models such as $AR(p)$. They provide a powerful approach for prediction in financial time series and share parameters across time. Convolution neural networks are useful as spectral transformations of spatial and temporal data and generalize techniques such as wavelets, which use fixed basis functions. They share parameters across space. Finally, autoencoders are used to compress information and generalize principal component analysis.

Chapter Objectives

By the end of this chapter, the reader should expect to accomplish the following:

- Characterize RNNs as non-linear autoregressive models and analyze their stability;
- Understand how gated recurrent units and long short-term memory architectures give a dynamic autoregressive model with variable memory;
- Characterize CNNs as regression, classification, and time series regression of filtered data;
- Understand principal component analysis for dimension reduction;
- Formulate a linear autoencoder and extract the principal components; and
- Understand how to build more complex networks by aggregating these different concepts.

The notebooks provided in the accompanying source code repository demonstrate many of the methods in this chapter. See Appendix “Python Notebooks” for further details.

2 Recurrent Neural Networks

Recall that if the data $\mathcal{D} := \{x_t, y_t\}_{t=1}^N$ is auto-correlated observations of X and Y at times $t = 1, \dots, N$, then the prediction problem can be expressed as a sequence prediction problem: construct a non-linear times series predictor, \hat{y}_{t+h} , of a response, y_{t+h} , using a high-dimensional input matrix of T length sub-sequences \mathcal{X}_t :

$$\hat{y}_{t+h} = f(\mathcal{X}_t) \text{ where } \mathcal{X}_t := seq_{T,t}(X) = (x_{t-T+1}, \dots, x_t),$$

where x_{t-j} is a j th lagged observation of x_t , $x_{t-j} = L^j[x_t]$, for $j = 0, \dots, T - 1$. Sequence learning, then, is just a composition of a non-linear map and a vectorization of the lagged input variables. If the data is i.i.d., then no sequence is needed (i.e., $T = 1$), and we recover a feedforward neural network.

Recurrent neural networks (RNNs) are times series methods or sequence learners which have achieved much success in applications such as natural language understanding, language generation, video processing, and many other tasks (Graves 2012). There are many types of RNNs—we will just concentrate on simple RNN models for brevity of notation. Like multivariate structural autoregressive models, RNNs apply an autoregressive function $f_{W^{(1)}, b^{(1)}}^{(1)}(X_t)$ to each input sequence X_t , where T denotes the look back period at each time step—the maximum number of lags. However, rather than directly imposing an autocovariance structure, a RNN provides a flexible functional form to directly model the predictor, \hat{Y} .

As illustrated in Fig. 8.1, this simple RNN is an unfolding of a single hidden layer neural network (a.k.a. Elman network (Elman 1991)) over all time steps in the sequence, $j = 0, \dots, T - 1$. For each time step, j , this function $f_{W^{(1)}, b^{(1)}}^{(1)}(X_{t,j})$ generates a hidden state z_{t-j} from the current input x_t and the previous hidden state z_{t-1} and $X_{t,j} = \text{seq}_{T,t-j}(X) \subset X_t$:

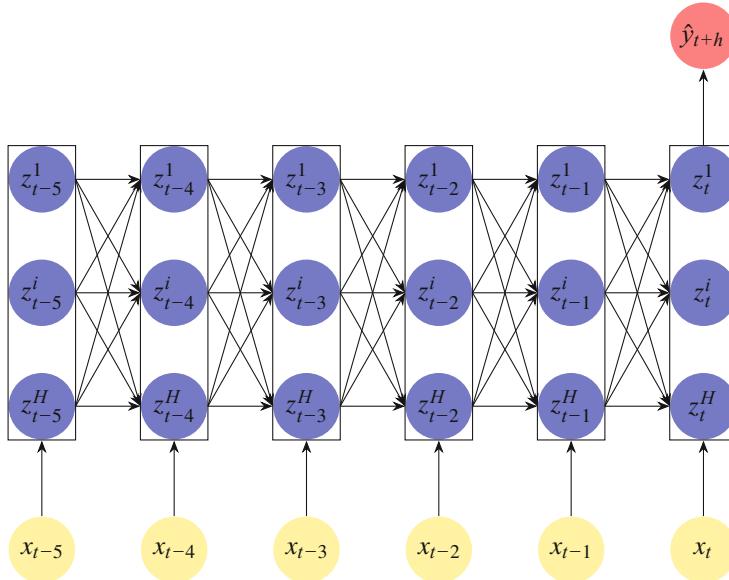


Fig. 8.1 An illustrative example of a recurrent neural network with one hidden layer, “unfolded” over a sequence of six time steps. Each input x_t is in the sequence X_t . The hidden layer contains H units and the i th output at time step t is denoted by z_t^i . The connections between the hidden units are recurrent and are weighted by the matrix $W_z^{(1)}$. At the last time step t , the hidden units connect to a single unit output layer with continuous \hat{y}_{t+h}

$$\begin{aligned} \text{response: } \hat{y}_{t+h} &= f_{W^{(2)}, b^{(2)}}^{(2)}(z_t) := \sigma^{(2)}(W^{(2)}z_t + b^{(2)}), \\ \text{hidden states: } z_{t-j} &= f_{W^{(1)}, b^{(1)}}^{(1)}(\mathcal{X}_{t,j}) \\ &:= \sigma^{(1)}(W_z^{(1)}z_{t-j-1} + W_x^{(1)}x_{t-j} + b^{(1)}), j \in \{T-1, \dots, 0\}, \end{aligned}$$

where $\sigma^{(1)}$ is an activation function such as $\tanh(x)$, and $\sigma^{(2)}$ is either a softmax function or identity map depending on whether the response is categorical or continuous, respectively. The connections between the extremal inputs x_t and the H hidden units are weighted by the time invariant matrix $W_x^{(1)} \in \mathbb{R}^{H \times P}$. The recurrent connections between the H hidden units are weighted by the time invariant matrix $W_z^{(1)} \in \mathbb{R}^{H \times H}$. Without such a matrix, the architecture is simply a single-layered feedforward network without memory—each independent observation x_t is mapped to an output \hat{y}_t using the same hidden layer.

The sets $W^{(1)} = (W_x^{(1)}, W_z^{(1)})$ refer to the input and recurrence weights. $W^{(2)}$ denotes the weights tied to the output of the H hidden units at the last time step, z_t , and the output layer. If the response is a continuous vector, $Y \in \mathbb{R}^M$, then $W^{(2)} \in \mathbb{R}^{M \times H}$. If the response is categorical, with K states, then $W^{(2)} \in \mathbb{R}^{K \times H}$. The number of hidden units determines the degree of non-linearity in the model and must be at least the dimensionality of the input p . In our experiments the hidden layer is generally under a hundred units, but can increase to thousands in higher dimensional datasets.

There are a number of issues in the RNN design. How many times should the network being unfolded? How many hidden neurons H in the hidden layer? How to perform “variable selection”? The answer to the first question lies in tests for autocorrelation of the data—the sequence length needed in a RNN can be determined by the largest significant lag in an estimated “partial autocorrelation” function. The answer to the second is no different to the problem of how to choose the number of hidden neurons in a MLP—the bias–variance tradeoff being the most important consideration. And indeed the third question is also closely related to the problem of choosing features in a MLP. One can take a principled approach to feature selection, first identifying a subset of features using a Granger-causality test, or the “laissez-machine” approach of including all potentially relevant features and allowing auto-shrinkage to determine the most important weights are more aligned with contemporary experimental design in machine learning. One important caveat on feature selection for RNNs is that each feature must be a time series and therefore exhibit autocorrelation.

We shall begin with a simple, univariate, example to illustrate how a RNN, without activation, is a $AR(p)$ time series model.

Example 8.1 RNNs as Non-linear AR(p) Models

Consider the simplest case of a RNN with one hidden unit, $H = 1$, no activation function, and the dimensionality of the input vector is $P = 1$. Suppose further that $W_z^{(1)} = \phi_z$, $|\phi_z| < 1$, $W_x^{(1)} = \phi_x$, $W_y = 1$, $b_h = 0$ and $b_y = \mu$. Then we can show that $f_{W_z^{(1)}, b^{(1)}}^{(1)}(X_t)$ is of an autoregressive, $AR(p)$, model of order p with geometrically decaying autoregressive coefficients $\phi_i = \phi_x \phi_z^{i-1}$:

$$\begin{aligned} z_{t-p} &= \phi_x x_{t-p} \\ z_{t-T+2} &= \phi_z z_{t-T+1} + \phi_x x_{t-T+2} \\ &\dots = \dots \\ z_{t-1} &= \phi_z z_{t-2} + \phi_x x_{t-1} \\ \hat{x}_t &= z_{t-1} + \mu \end{aligned}$$

then

$$\begin{aligned} \hat{x}_t &= \mu + \phi_x (L + \phi_z L^2 + \dots + \phi_z^{p-1} L^p)[x_t] \\ &= \mu + \sum_{i=1}^p \phi_i x_{t-i} \end{aligned}$$

This special type of autoregressive model \hat{x}_t is “stable” and the order can be identified through autocorrelation tests on X such as the Durbin–Watson, Ljung–Box, or Box–Pierce tests. Note that if we modify the architecture so that the recurrence weights $W_{z,i}^{(1)} = \phi_{z,i}$ are lag dependent then the unactivated hidden layer is

$$z_{t-i} = \phi_{z,i} z_{t-i-1} + \phi_x x_{t-i} \quad (8.1)$$

which gives

$$\hat{x}_t = \mu + \phi_x (L + \phi_{z,1} L^2 + \dots + \prod_{i=1}^{p-1} \phi_{z,i} L^p)[x_t], \quad (8.2)$$

and thus the weights in this $AR(p)$ model are $\phi_j = \phi_x \prod_{i=1}^{j-1} \phi_{z,i}$ which allows a more flexible presentation of the autocorrelation structure than the plain RNN—which is limited to geometrically decaying weights. Note that a linear RNN with infinite number of lags and no bias corresponds to an exponential smoother, $z_t = \alpha x_t + (1 - \alpha) z_{t-1}$ when $W_z = 1 - \alpha$, $W_x = \alpha$, and $W_y = 1$.

The generalization of a linear RNN from $AR(p)$ to $VAR(p)$ is trivial and can be written as

$$\hat{x}_t = \boldsymbol{\mu} + \sum_{j=1}^p \phi_j x_{t-j}, \quad \phi_j := W^{(2)} (W_z^{(1)})^{j-1} W_x^{(1)}, \quad \boldsymbol{\mu} := W^{(2)} \sum_{j=1}^p (W_z^{(1)})^{j-1} b^{(1)} + b^{(2)}, \quad (8.3)$$

where the square matrix $\phi_j \in \mathbb{R}^{P \times P}$ and bias vector $\boldsymbol{\mu} \in \mathbb{R}^P$.

2.1 RNN Memory: Partial Autocovariance

Generally, with non-linear activation, it is more difficult to describe the RNN as a classical model. However, the partial autocovariance function provides some additional insight here. Let us first consider a RNN(1) process. The lag-1 partial autocovariance is

$$\tilde{\gamma}_1 = \mathbb{E}[y_t - \mu, y_{t-1} - \mu] = \mathbb{E}[\hat{y}_t + \epsilon_t - \mu, y_{t-1} - \mu], \quad (8.4)$$

and using the RNN(1) model with, for simplicity, a single recurrence weight, ϕ :

$$\hat{y}_t = \sigma(\phi y_{t-1}) \quad (8.5)$$

gives

$$\tilde{\gamma}_1 = \mathbb{E}[\sigma(\phi y_{t-1}) + \epsilon_t - \mu, y_{t-1} - \mu] = \mathbb{E}[y_{t-1} \sigma(\phi y_{t-1})], \quad (8.6)$$

where we have assumed $\mu = 0$ in the second part of the expression. Checking that we recover the AR(1) covariance, set $\sigma := Id$ so that

$$\tilde{\gamma}_1 = \phi \mathbb{E}[y_{t-1}^2] = \phi \mathbb{V}[y_{t-1}]. \quad (8.7)$$

Continuing with the lag-2 autocovariance gives:

$$\tilde{\gamma}_2 = \mathbb{E}[y_t - P(y_t | y_{t-1}), y_{t-2} - P(y_{t-2} | y_{t-1})], \quad (8.8)$$

and $P(y_t | y_{t-1})$ is approximated by the RNN(1):

$$\hat{y}_t = \sigma(\phi y_{t-1}). \quad (8.9)$$

Substituting $y_t = \hat{y}_t + \epsilon_t$ into the above gives

$$\tilde{\gamma}_2 = \mathbb{E}[\epsilon_t, y_{t-2} - P(y_{t-2} | y_{t-1})]. \quad (8.10)$$

Approximating $P(y_{t-2} \mid y_{t-1})$ with the backward RNN(1)

$$\hat{y}_{t-2} = \sigma(\phi(\hat{y}_{t-1} + \epsilon_{t-1})), \quad (8.11)$$

we see, crucially, that \hat{y}_{t-2} depends on ϵ_{t-1} but not on ϵ_t . $y_{t-2} - P(y_{t-2} \mid y_{t-1})$, hence depends on $\{\epsilon_{t-1}, \epsilon_{t-2}, \dots\}$. Thus we have that $\tilde{\gamma}_2 = 0$.

As a counterexample, consider the lag-2 partial autocovariance of the RNN(2) process

$$\hat{y}_{t-2} = \sigma(\phi\sigma(\phi(\hat{y}_t + \epsilon_t) + \epsilon_{t-1})), \quad (8.12)$$

which depends on ϵ_t and hence the lag-2 partial autocovariance is not zero.

It is easy to show that the partial autocorrelation $\tilde{\tau}_s = 0, s > p$ and, thus, like the AR(p) process, the partial autocorrelation function for a RNN(p) has a cut-off at p lags. The partial autocorrelation function is independent of time. Such a property can be used to identify the order of the RNN model from the estimated PACF.

2.2 Stability

We can generalize the stability constraint on AR(p) models presented in Sect. 2.3 to RNNs by considering the RNN(1) model:

$$y_t = \Phi^{-1}(L)[\epsilon_t] = (1 - \sigma(W_z L + b))^{-1} [\epsilon_t], \quad (8.13)$$

where we have set $W_y = 1$ and $b_y = 0$ without loss of generality, and dropped the superscript (1) for ease of notation. Expressing this as an infinite dimensional non-linear moving average model

$$y_t = \frac{1}{1 - \sigma(W_z L + b)} [\epsilon_t] = \sum_{j=0}^{\infty} \sigma^j (W_z L + b) [\epsilon_t], \quad (8.14)$$

and the infinite sum will be stable when the $\sigma^j(\cdot)$ terms do not grow with j , i.e. $|\sigma| \leq 1$ for all values of ϕ and y_{t-1} . In particular, the choice \tanh satisfies the requirement on σ . For higher order models, we follow an induction argument and show first that for a RNN(2) model we obtain

$$\begin{aligned} y_t &= \frac{1}{1 - \sigma(W_z \sigma(W_z L^2 + b) + W_x L + b)} [\epsilon_t] \\ &= \sum_{j=0}^{\infty} \sigma^j (W_z \sigma(W_z L^2 + b) + W_x L + b) [\epsilon_t], \end{aligned}$$

which again is stable if $|\sigma| \leq 1$ and it follows for any model order that the stability condition holds.

It follows that lagged unit impulses of the *data* strictly decay with the order of the lag when $|\sigma| \leq 1$. Again by induction, at lag 1, the output from the hidden layer is

$$z_t = \sigma(W_z \mathbf{1} + W_x \mathbf{0} + b) = \sigma(W_z \mathbf{1} + b). \quad (8.15)$$

The absolute value of each component of the hidden variable under a unit vector impulse at lag 1 is strictly less than 1:

$$|z_t|_j = |\sigma(W_z \mathbf{1} + b)|_j < 1, \quad (8.16)$$

if $|\sigma(x)| \leq 1$ and each element of $W_z \mathbf{1} + b$ is finite. Additionally if σ is strictly monotone increasing, then $|z_t|_j$ under a lag two unit innovation is strictly less than $|z_t|_j$ under a lag one unit innovation

$$|\sigma(W_z \mathbf{1}) + b|_j > |\sigma(W_z \sigma(W_z \mathbf{1} + b) + b)|_j. \quad (8.17)$$

The implication of this stability result is the reassuring attribute that past random disturbances decay in the model and the effect of lagged data becomes less relevant to the model output with increasing lag.

2.3 Stationarity

For completeness, we mention in passing the extension of stationarity analysis in Sect. 2.4 to RNNs. The linear univariate RNN(p), considered above, has a companion matrix of the form

$$C := \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 \\ \phi^{-p} & -\phi^{-p+1} & \dots & -\phi^{-2} & -\phi^{-1}, \end{pmatrix} \quad (8.18)$$

and it turns out that for $\phi \neq 0$ this model is non-stationary. We can hence rule out the choice of a linear activation since this would leave us with a linear RNN. Hence, it appears that some non-linear activation is necessary for the model to be stationary, but we cannot use the Cayley–Hamilton theorem to prove stationarity.

Half-Life

Suppose that the output of the RNN is in \mathbb{R}^d . The half-life of the lag is the smallest number of function compositions, k , of $\tilde{\sigma}(x) := \sigma(W_z x + b)$ with itself such that the normalized j th output is

$$r_j^{(k)} = \frac{(W_y \tilde{\sigma} \circ_1 \tilde{\sigma} \circ_2 \cdots \circ_{k-1} \tilde{\sigma}(\mathbf{1}) + b_y)_j}{(W_y \tilde{\sigma}(\mathbf{1}) + b_y)_j} \leq 0.5, k \geq 2, \forall j \in \{1, \dots, d\}. \quad (8.19)$$

Note the output has been normalized so that the lag-1 unit impulse ensures that the ratio, $r_j^{(1)} = 1$ for each j . This modified definition exists to account for the effects of the activation function and the semi-affine transformation which are not present in AR(p) model. In general, there is no guarantee that the half-life is finite but we can find parameter values for which the half-life can be found. For example, suppose for simplicity that a univariate RNN is given by $\hat{x}_t = z_{t-1}$ and

$$z_t = \sigma(z_{t-1} + x_t).$$

Then the lag-1 impulse is $\hat{x}_t = \tilde{\sigma}(\mathbf{1}) = \sigma(\mathbf{0} + \mathbf{1})$, the lag-2 impulse is $\hat{x}_t = \sigma(\sigma(\mathbf{1}) + \mathbf{0}) = \tilde{\sigma} \circ \tilde{\sigma}(\mathbf{1})$, and so on. If $\sigma(x) := \tanh(x)$ and we normalize over the output from the lag-1 impulse to give the values in Table 8.1.

Table 8.1 The half-life characterizes the memory decay of the architecture by measuring the number of periods before a lagged unit impulse has at least half of its effect at lag 1. The calculation of the half-life involves nested composition of the recursion relation for the hidden layer until $r_j^{(k)}$ is less than a half. The calculations are repeated for each j , hence the half-life may vary depending on the component of the output. In this example, the half-life of the univariate RNN is 9 periods

Lag k	$r^{(k)}$
1	1.000
2	0.843
3	0.744
4	0.673
5	0.620
6	0.577
7	0.543
8	0.514
9	0.489

? Multiple Choice Question 1

Which of the following statements are true:

- a. An augmented Dickey–Fuller test can be applied to time series to determine whether they are covariance stationary.

- b. The estimated partial autocorrelation of a covariance stationary time series can be used to identify the design sequence length in a plain recurrent neural network.
 - c. Plain recurrent neural networks are guaranteed to be stable, namely lagged unit impulses decay over time.
 - d. The Ljung–Box test is used to test whether the fitted model residual error is auto-correlated.
 - e. The half-life of a lag-1 unit impulse is the number of lags before the impulse has half its effect on the model output.
-

2.4 Generalized Recurrent Neural Networks (GRNNs)

Classical RNNs, such as those described above, treat the error as homoscedastic—that is, the error is i.i.d. We mention in passing that we can generalize RNNs to heteroscedastic models by modifying the loss function to the squared Mahalanobis length of the residual vector. Such an approach is referred to here as generalized recurrent neural networks (GRNNs) and is mentioned briefly here with the caveat that the field of machine learning in econometrics is nascent and therefore incomplete and such a methodology, while appealing from a theoretic perspective, is not yet proven in practice. Hence the purpose of this subsection is simply to illustrate how more complex models can be developed which mirror some of the developments in parametric econometrics.

In its simplest form, we solve a weighted least squares minimization problem using data, \mathcal{D}_t :

$$\underset{W, b}{\text{minimize}} \quad f(W, b) + \lambda \phi(W, b), \quad (8.20)$$

$$\mathcal{L}_\Sigma(Y, \hat{Y}) := (Y - \hat{Y})^T \Sigma^{-1} (Y - \hat{Y}), \quad \Sigma_{tt} = \sigma_t^2, \quad \Sigma_{tt'} = \rho_{tt'} \sigma_t \sigma'_{t'}, \quad (8.21)$$

$$f(W, b) = \frac{1}{T} \sum_{t=1}^T \mathcal{L}_\Sigma(y_t, \hat{y}_t), \quad (8.22)$$

where $\Sigma := \mathbb{E}[\epsilon \epsilon^T | X_t]$ is the conditional covariance matrix of the residual error and $\phi(W, b)$ is a regularization penalty term.

The conditional covariance matrix of the error must be estimated. This is performed as follows using the notation $()^T$ to denote the transpose of a vector and $()'$ to denote model parameters fitted under heteroscedastic error.

- 1) For each $t = 1, \dots, T$, estimate the residual error over the training set, $\epsilon_t \in \mathbb{R}^N$, using the standard (unweighted) loss function to find the weights, \hat{W}_t , and biases, \hat{b}_t where the error is

$$\epsilon_t = \mathbf{y}_t - F_{\hat{W}_t, \hat{b}_t}(\mathcal{X}_t). \quad (8.23)$$

2) The sample conditional covariance matrix $\hat{\Sigma}$ is estimated accordingly:

$$\hat{\Sigma} = \frac{1}{T-1} \sum_{i=1}^T \epsilon_i \epsilon_i^T. \quad (8.24)$$

3) Perform the generalized least squares minimization using Eq. 8.20 to obtain a fitted heteroscedastic neural network model, with refined error

$$\epsilon'_t = \mathbf{y}_t - F_{\hat{W}'_t, \hat{b}'_t}(\mathcal{X}_t). \quad (8.25)$$

The fitted GRNN $F_{\hat{W}'_t, \hat{b}'_t}$ can then be used for forecasting without any further modification. The effect of the sample covariance matrix is to adjust the importance of the observation in the training set, based on the variance of its error and the error correlation. Such an approach can be broadly viewed as a RNN analogue of how GARCH models extend AR models. Of course, GARCH models treat the error distribution as parametric and provide a recurrence relation for forecasting the conditional volatility. In contrast, GRNNs rely on the empirical error distribution and do not forecast the conditional volatility. However, a separate regression could be performed over diagonals of the empirical conditional volatility Σ by using time series cross-validation.

3 Gated Recurrent Units

The extension of RNNs to dynamical time series models rests on extending foundational concepts in time series analysis. We begin by considering a smoothed RNN with hidden state \hat{h}_t . Such a RNN is almost identical to a plain RNN, but with an additional scalar smoothing parameter, α , which provides the network with “long memory.”

3.1 α -RNNs

Let us consider a univariate α -RNN(p) model in which the smoothing parameter is fixed:

$$\hat{y}_{t+1} = W_y \hat{h}_t + b_y, \quad (8.26)$$

$$\hat{h}_t = \sigma(U_h \tilde{h}_{t-1} + W_h y_t + b_h), \quad (8.27)$$

$$\tilde{h}_t = \alpha \hat{h}_{t-1} + (1 - \alpha) \tilde{h}_{t-1}, \quad (8.28)$$

with the starting condition in each sequence, $\hat{h}_{t-p+1} = y_{t-p+1}$. This model augments the plain RNN by replacing \hat{h}_{t-1} in the hidden layer with an exponentially smoothed hidden state \tilde{h}_{t-1} . The effect of the smoothing is to provide infinite memory when $\alpha \neq 1$. For the special case when $\alpha = 1$, we recover the plain RNN with short memory of length p .

We can easily study this model by simplifying the parameterization and considering the unactivated case. Setting $b_y = b_h = 0$, $U_h = W_h = \phi$ and $W_y = 1$:

$$\hat{y}_{t+1} = \hat{h}_t, \quad (8.29)$$

$$= \phi(\tilde{h}_{t-1} + y_t), \quad (8.30)$$

$$= \phi(\alpha \hat{h}_{t-1} + (1 - \alpha) \tilde{h}_{t-2} + y_t). \quad (8.31)$$

Without loss of generality, consider $p = 2$ lags in the model so that $\hat{h}_{t-1} = \phi y_{t-1}$. Then

$$\hat{h}_t = \phi(\alpha \phi y_{t-1} + (1 - \alpha) \tilde{h}_{t-2} + y_t) \quad (8.32)$$

and the model can be written in the simpler form

$$\hat{y}_{t+1} = \phi_1 y_t + \phi_2 y_{t-1} + \phi(1 - \alpha) \tilde{h}_{t-2}, \quad (8.33)$$

with autoregressive weights $\phi_1 := \phi$ and $\phi_2 := \alpha\phi^2$. We now see, in comparison with an AR(2) model, that there is an additional term which vanishes when $\alpha = 1$ but provides infinite memory to the model since \tilde{h}_{t-2} depends on y_0 , the first observation in the whole time series, not just the first observation in the sequence. The α -RNN model can be trained by treating α as a hyperparameter. The choice to fix α is obviously limited to stationary time series. We can extend the model to non-stationary time series by using a dynamic version of exponential smoothing.

3.1.1 Dynamic α_t -RNNs

Dynamic exponential smoothing is a time-dependent, convex, combination of the smoothed output, \tilde{y}_t , and the observation y_t :

$$\tilde{y}_{t+1} = \alpha_t y_t + (1 - \alpha_t) \tilde{y}_t, \quad (8.34)$$

where $\alpha_t \in [0, 1]$ denotes the dynamic smoothing factor which can be equivalently written in the one-step-ahead forecast of the form

$$\tilde{y}_{t+1} = \tilde{y}_t + \alpha_t(y_t - \tilde{y}_t). \quad (8.35)$$

Hence the smoothing can be viewed as a form of dynamic forecast error correction; When $\alpha_t = 0$, the forecast error is ignored and the smoothing merely repeats the current hidden state \tilde{h}_t to the effect of the model losing its memory. When $\alpha_t = 1$, the forecast error overwrites the current hidden state \tilde{h}_t .

The smoothing can also be viewed a weighted sum of the lagged observations, with lower or equal weights, $\alpha_{t-s} \prod_{r=1}^s (1 - \alpha_{t-r+1})$ at the lag $s \geq 1$ past observation, y_{t-s} :

$$\tilde{y}_{t+1} = \alpha_t y_t + \sum_{s=1}^{t-1} \alpha_{t-s} \prod_{r=1}^s (1 - \alpha_{t-r+1}) y_{t-s} + \prod_{r=0}^{t-1} (1 - \alpha_{t-r}) \tilde{y}_1, \quad (8.36)$$

where the last term is a time-dependent constant and typically we initialize the exponential smoother with $\tilde{y}_1 = y_1$. Note that for any $\alpha_{t-r+1} = 1$, the prediction \tilde{y}_{t+1} will have no dependency on all lags $\{y_{t-s}\}_{s \geq r}$. The model simply forgets the observations at or beyond the r th lag.

In the special case when the smoothing is constant and equal to $1 - \alpha$, then the above expression simplifies to

$$\tilde{y}_{t+1} = \alpha \Phi(L)^{-1} y_t, \quad (8.37)$$

or equivalently written as a AR(1) process in \tilde{y}_{t+1} :

$$\Phi(L) \tilde{y}_{t+1} = \alpha y_t, \quad (8.38)$$

for the linear operator $\Phi(z) := 1 + (\alpha - 1)z$ and where L is the lag operator.

3.2 Neural Network Exponential Smoothing

Let us suppose now that instead of smoothing the observed time series $\{y_s\}_{s \leq 1}$, we instead smooth a hidden vector \hat{h}_t with $\hat{\alpha}_t \in [0, 1]^H$ to give a filtered time series

$$\tilde{h}_t = \hat{\alpha}_t \circ \hat{h}_t + (1 - \hat{\alpha}_t) \circ \tilde{h}_{t-1}, \quad (8.39)$$

where \circ denotes the Hadamard product between vectors. This smoothing is a vectorized form of the above classical setting, only here we note that when $(\hat{\alpha}_t)_i = 1$, the i th component of the hidden variable is unmodified and the past filtered hidden variable is forgotten. On the other hand, when the $(\hat{\alpha}_t)_i = 0$, the i th component of the hidden variable is obsolete, instead setting the current filtered hidden variable to its past value. The smoothing in Eq. 8.39 can be viewed then as updating long-term memory, maintaining a smoothed hidden state variable as the memory through a convex combination of the current hidden variable and the previous smoothed hidden variable.

The hidden variable is given by the semi-affine transformation:

$$\hat{h}_t = \sigma(U_h \tilde{h}_{t-1} + W_h x_t + b_h) \quad (8.40)$$

which in turns depends on the previous smoothed hidden variable. Substituting Eq. 8.40 into Eq. 8.39 gives a function of \tilde{h}_{t-1} and x_t :

$$\tilde{h}_t = g(\tilde{h}_{t-1}, x_t; \alpha) := \hat{\alpha}_t \circ \sigma(U_h \tilde{h}_{t-1} + W_h x_t + b_h) + (1 - \hat{\alpha}_t) \circ \tilde{h}_{t-1}. \quad (8.41)$$

We see that when $\alpha_t = 0$, the smoothed hidden variable \tilde{h}_t is not updated by the input x_t . Conversely, when $\alpha_t = 1$, we observe that the hidden variable locally behaves like a non-linear autoregressive series. Thus the smoothing parameter can be viewed as the sensitivity of the smoothed hidden state to the input x_t .

The challenge becomes how to determine dynamically how much error correction is needed. GRUs address this problem by learning $\hat{\alpha} = F_{(W_\alpha, U_\alpha, b_\alpha)}(X)$ from the input variables with a plain RNN parameterized by weights and biases $(W_\alpha, U_\alpha, b_\alpha)$. The one-step-ahead forecast of the smoothed hidden state, \tilde{h}_t , is the filtered output of another plain RNN with weights and biases (W_h, U_h, b_h) . Putting this together gives the following $\alpha - t$ model (simple GRU):

$$\text{smoothing : } \tilde{h}_t = \hat{\alpha}_t \circ \hat{h}_t + (1 - \hat{\alpha}_t) \circ \tilde{h}_{t-1} \quad (8.42)$$

$$\text{smoother update : } \hat{\alpha}_t = \sigma^{(1)}(U_\alpha \tilde{h}_{t-1} + W_\alpha x_t + b_\alpha) \quad (8.43)$$

$$\text{hidden state update : } \hat{h}_t = \sigma(U_h \tilde{h}_{t-1} + W_h x_t + b_h), \quad (8.44)$$

where $\sigma^{(1)}$ is a sigmoid or Heaviside function and σ is any activation function. Figure 8.2 shows the response of a α_t -RNN when the input consists of two unit impulses. For simplicity, the sequence length is assumed to be 3 (i.e., the RNN has a memory of 3 lags), the biases are set to zero, all the weights are set to one, and $\sigma(x) := \tanh(x)$. Note that the weights have not been fitted here, we are merely observing the effect of smoothing on the hidden state for the simplest choice of parameter values. The RNN loses memory of the unit impulse after three lags, whereas the RNNs with smooth hidden states maintain memory of the first unit impulse even when the second unit impulse arrives. The difference between the dynamically smoothed RNN (the α_t -RNN) and α -RNN with a fixed smoothing parameter appears insignificant. Keep in mind however that the dynamical smoothing model has much more flexibility in how it controls the sensitivity of the smoothing to the unit impulses.

In the above α_t -RNN, there is no means to directly occasionally forget the memory. This is because the hidden variables update equation always depends on the previous smoothed hidden state, unless $U_h = 0$. However, it can be expected that the fitted recurrence weight \hat{U}_h will not in general be zero and thus the model is without a “hard reset button.”

GRUs also have the capacity to entirely reset the memory by adding an additional reset variable:

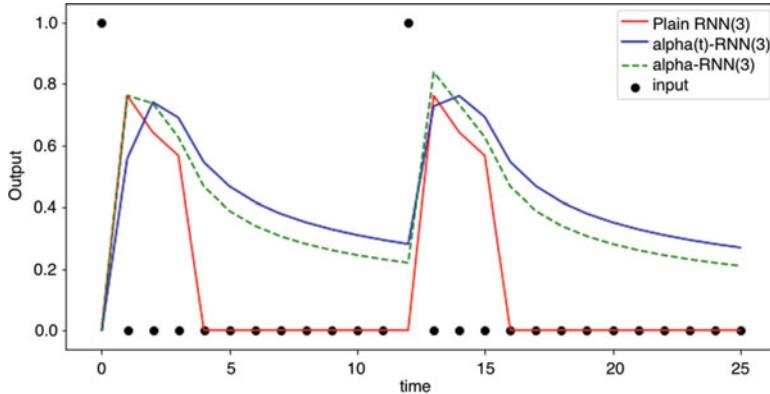


Fig. 8.2 An illustrative example of the response of an α_t -RNN and comparison with a plain RNN and a RNN with an exponentially smoothed hidden state, under a constant α (α -RNN). The RNN(3) model loses memory of the unit impulse after three lags, whereas the α -RNN(3) models maintain memory of the first unit impulse even when the second unit impulse arrives. The difference between the α_t -RNN (the toy GRU) and the α -RNN appears insignificant. Keep in mind however that the dynamical smoothing model has much more flexibility in how it controls the sensitivity of the smoothing to the unit impulses

$$\text{smoothing : } \tilde{h}_t = \hat{\alpha}_t \circ \hat{h}_t + (1 - \hat{\alpha}_t) \circ \tilde{h}_{t-1} \quad (8.45)$$

$$\text{smoother update : } \hat{\alpha}_t = \sigma^{(1)}(U_\alpha \tilde{h}_{t-1} + W_\alpha x_t + b_\alpha) \quad (8.46)$$

$$\text{hidden state update : } \hat{h}_t = \sigma(U_h \hat{r}_t \circ \tilde{h}_{t-1} + W_h x_t + b_h) \quad (8.47)$$

$$\text{reset update : } \hat{r}_t = \sigma^{(1)}(U_r \tilde{h}_{t-1} + W_r x_t + b_r). \quad (8.48)$$

The effect of introducing a reset, or switch, \hat{r}_t , is to forget the dependence of \hat{h}_t on the smoothed hidden state. Effectively, we turn the update for \hat{h}_t from a plain RNN to a FFN and entirely neglect the recurrence. The recurrence in the update of \hat{h}_t is thus dynamic. It may appear that the combination of a reset and adaptive smoothing is redundant. But remember that $\hat{\alpha}_t$ effects the level of error correction in the update of the smoothed hidden state, \tilde{h}_t , whereas \hat{r}_t adjusts the level of recurrence in the unsmoothed hidden state \hat{h}_t . Put differently, $\hat{\alpha}_t$ by itself cannot disable the memory in the smoothed hidden state (internal memory), whereas \hat{r}_t in combination with $\hat{\alpha}_t$ can. More precisely, when $\alpha_t = 1$ and $\hat{r}_t = 0$, $\hat{h}_t = \tilde{h}_t = \sigma(W_h x_t + b_h)$ which is reset to the latest input, x_t , and the GRU is just a FFNN. Also, when $\alpha_t = 1$ and $\hat{r}_t > 0$, a GRU acts like a plain RNN. Thus a GRU can be seen as a more general architecture which is capable of being a FFNN or a plain RNN under certain parameter values.

These additional layers (or cells) enable a GRU to learn extremely complex long-term temporal dynamics that a plain RNN is not capable of. The price to pay for this flexibility is the additional complexity of the model. Clearly, one must choose whether to opt for a simpler model, such as an α_t -RNN, or use a GRU. Lastly, we

comment in passing that in the GRU, as in a RNN, there is a final feedforward layer to transform the (smoothed) hidden state to a response:

$$\hat{y}_t = W_Y \tilde{h}_t + b_Y. \quad (8.49)$$

3.3 Long Short-Term Memory (LSTM)

The GRU provides a gating mechanism for propagating a smoothed hidden state—a long-term memory—which can be overridden and even turn the GRU into a plain RNN (with short memory) or even a memoryless FFN. More complex models using hidden units with varying connections within the memory unit have been proposed in the engineering literature with empirical success (Hochreiter and Schmidhuber 1997; Gers et al. 2001; Zheng et al. 2017). LSTMs are similar to GRUs but have a separate (cell) memory, C_t , in addition to a hidden state h_t . LSTMs also do not require that the memory updates are a convex combination. Hence they are more general than exponential smoothing. The mathematical description of LSTMs is rarely given in an intuitive form, but the model can be found in, for example, Hochreiter and Schmidhuber (1997).

The cell memory is updated by the following expression involving a forget gate, $\hat{\alpha}_t$, an input gate \hat{z}_t , and a cell gate \hat{c}_t

$$c_t = \hat{\alpha}_t \circ c_{t-1} + \hat{z}_t \circ \hat{c}_t. \quad (8.50)$$

In the language of LSTMs, the triple $(\hat{\alpha}_t, \hat{r}_t, \hat{z}_t)$ are, respectively, referred to as the forget gate, output gate, and input gate. Our change of terminology is deliberate and designed to provide more intuition and continuity with GRUs and econometrics. We note that in the special case when $\hat{z}_t = 1 - \hat{\alpha}_t$ we obtain a similar exponential smoothing expression to that used in the GRU. Beyond that, the role of the input gate appears superfluous and difficult to reason with using time series analysis. Likely it merely arose from a contextual engineering model; however, it is tempting to speculate how the additional variable provides the LSTM with a more elaborate representation of complex temporal dynamics.

When the forget gate, $\hat{\alpha}_t = 0$, then the cell memory depends solely on the cell memory gate update \hat{c}_t . By the term $\hat{\alpha}_t \circ c_{t-1}$, the cell memory has long-term memory which is only forgotten beyond lag s if $\hat{\alpha}_{t-s} = 0$. Thus the cell memory has an adaptive autoregressive structure.

The extra “memory,” treated as a hidden state and separate from the cell memory, is nothing more than a Hadamard product:

$$h_t = \hat{r}_t \circ \tanh(c_t), \quad (8.51)$$

which is reset if $\hat{r}_t = 0$. If $\hat{r}_t = 1$, then the cell memory directly determines the hidden state.

Thus the reset gate can entirely override the effect of the cell memory's autoregressive structure, without erasing it. In contrast, the GRU has one memory, which serves as the hidden state, and it is directly affected by the reset gate.

The reset, forget, input, and cell memory gates are updated by plain RNNs all depending on the hidden state h_t .

$$\text{Reset gate : } \hat{r}_t = \sigma(U_r h_{t-1} + W_r x_t + b_r) \quad (8.52)$$

$$\text{Forget gate : } \hat{\alpha}_t = \sigma(U_\alpha h_{t-1} + W_\alpha x_t + b_\alpha) \quad (8.53)$$

$$\text{Input gate : } \hat{z}_t = \sigma(U_z h_{t-1} + W_z x_t + b_z) \quad (8.54)$$

$$\text{Cell memory gate : } \hat{c}_t = \tanh(U_c h_{t-1} + W_c x_t + b_c). \quad (8.55)$$

Like the GRU, the LSTM can function as a short memory, plain RNN; just set $\alpha_t = 0$ in Eq. 8.50. However, the LSTM can also function as a coupling of FFNs; just set $\hat{r}_t = 0$ so that $h_t = 0$ and hence there is no recurrence structure in any of the gates. Both GRUs and LSTMs, even if the nomenclature does not suggest it, can model long- and short-term autoregressive memory. The GRU couple these through a smoothed hidden state variable. The LSTM separates out the long memory, stored in the cellular memory, but uses a copy of it, which may additionally be reset. Strictly speaking, the cellular memory has long-short autoregressive memory structure, so it would be misleading in the context of time series analysis to strictly discern the two memories as long and short (as the nomenclature suggests). The latter can be thought of as a truncated version of the former.

? Multiple Choice Question 2

Which of the following statements are true:

- a. A gated recurrent unit uses dynamic exponential smoothing to propagate a hidden state with infinite memory.
 - b. The gated recurrent unit requires that the data is covariance stationary.
 - c. Gated recurrent units are unconditionally stable, for any choice of activation functions and weights.
 - d. A GRU only has one memory, the hidden state, whereas a LSTM has an additional, cellular, memory.
-

4 Python Notebook Examples

The following Python examples demonstrate the application of RNNs and GRUs to financial time series prediction.

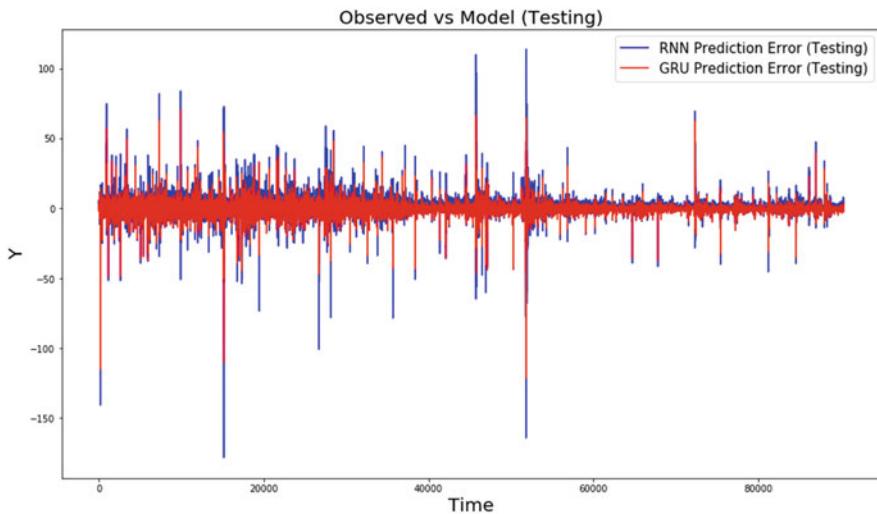


Fig. 8.3 A comparison of out-of-sample forecasting errors produced by a RNN and GRU trained on minute snapshots of Coinbase mid-prices

4.1 Bitcoin Prediction

`ML_in_Finance-RNNs-Bitcoin.ipynb` provides an example of how TensorFlow can be used to train and test RNNs for time series prediction. The example dataset is for predicting minute head mid-prices from minute snapshots of the USD value of Coinbase over 2018.

Statistical methods for stationarity and autocorrelation shall be used to characterize the data, identify the sequence length needed in the RNN, and to diagnose the model error. Here we accept the Null as the p-value is larger than 0.01 in absolute value and thus we cannot reject the ADF test at the 99% confidence level. Since plain RNNs are not suited to non-stationary time series modeling, we can use a GRU or LSTM to model non-stationarity data, since these models exhibit dynamic autocorrelation structure. Figure 8.3 compares the out-of-sample forecasting errors produced by a RNN and GRU. See the notebook for further details of the architecture and experiment.

4.2 Predicting from the Limit Order Book

The dataset is tick-by-tick, top of the limit order book, data such as mid-prices and volume weighted mid-prices (VWAP) collected from ZN futures. This dataset is heavily truncated for demonstration purposes and consists of 1033492 observations. The data has also been labeled to indicate whether the prices up-tick (1), remain

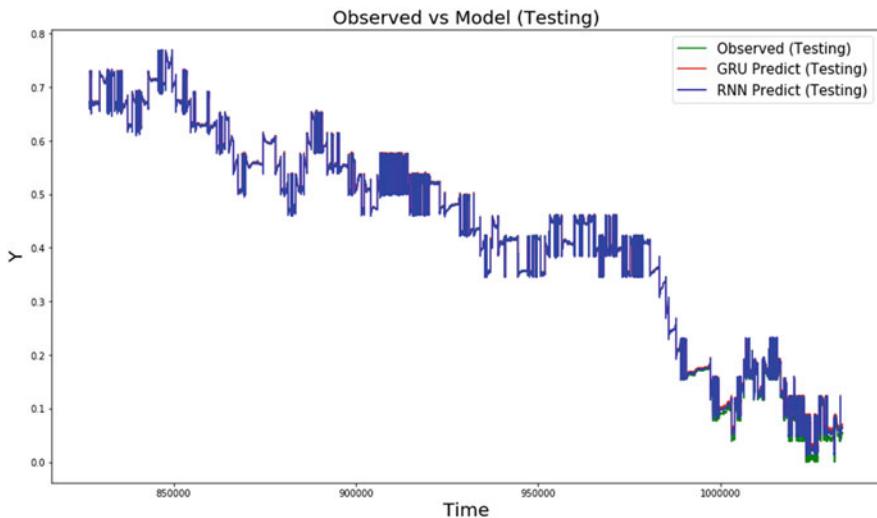


Fig. 8.4 A comparison of out-of-sample forecasting errors produced by a plain RNN and GRU trained on tick-by-tick smart prices of ZN futures

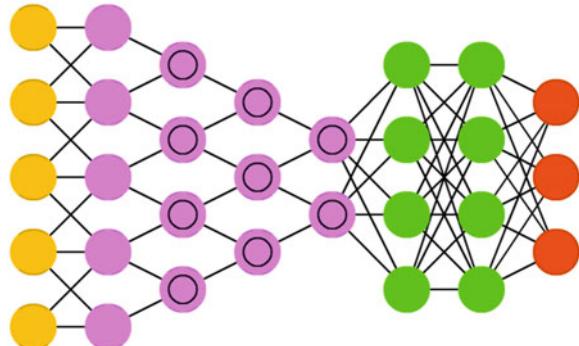
the same, or down-tick (-1) over the next tick. For demonstration purposes, the timestamps have been removed. In the simple forecasting experiment, we predict VWAPs (a.k.a. “smart prices”) from historical smart prices. Note that a classification experiment is also possible but not shown here.

The ADF test is performed over the first 200k observations as it is computationally intensive to apply it to the entire dataset. The ADF test statistic is -3.9706 and the p-value is smaller in absolute value than 0.01 and we thus reject the Null of the ADF test at the 99% confidence level in favor of the data being stationary (i.e., there are no unit roots). The Ljung–Box test is used to identify the number of lags needed in the model. A comparison of out-of-sample VWAP prices produced by a plain RNN and GRU is shown in Fig. 8.4. Because the data is stationary, we observe little advantage in using a GRU over a plain RNN. See `ML_in_Finance-RNNs-HFT.ipynb` for further details of the network architectures and experiment.

5 Convolutional Neural Networks

Convolutional neural networks (CNNs) are feedforward neural networks that can exploit local spatial structures in the input data. Flattening high-dimensional time series, such as limit order book depth histories, would require a very large number of weights in a feedforward architecture. CNNs attempt to reduce the network size by exploiting data locality (Fig. 8.5).

Fig. 8.5 Convolutional neural networks. Source: Van Veen, F. & Leijnen, S. (2019), “The Neural Network Zoo”, Retrieved from <https://www.asimovinstitute.org/neural-network-zoo>



Deep CNNs, with multiple consecutive convolutions followed by non-linear functions, have shown to be immensely successful in image processing (Krizhevsky et al. 2012). We can view convolutions as spatial filters that are designed to select a specific pattern in the data, for example, straight lines in an image. For this reason, convolution is frequently used for image processing, such as for smoothing, sharpening, and edge detection of images. Of course, in financial modeling, we typically have different spatial structures, such as the limit order book depths or the implied volatility surface of derivatives. However, the CNN has established its place in time series analysis too.

5.1 Weighted Moving Average Smoothers

A common technique in time series analysis and signal processing is to filter the time series. We have already seen exponential smoothing as a special case of a class of smoothers known as “weighted moving average (WMA)” smoothers. WMA smoothers take the form

$$\tilde{x}_t = \frac{1}{\sum_{i \in \mathcal{I}} w_i} \sum_{i \in \mathcal{I}} w_i x_{t-i}, \quad (8.56)$$

where \tilde{x}_t is the local mean of the time series. The weights are specified to emphasize or deemphasize particular observations of x_{t-i} in the span $|\mathcal{I}|$. Examples of well-known smoothers include the Hanning smoother $h(3)$:

$$\tilde{x}_t = (x_{t-1} + 2x_t + x_{t+1})/4. \quad (8.57)$$

Such smoothers have the effect of reducing noise in the time series. The moving average filter is a simple low pass finite impulse response (FIR) filter commonly used for regulating an array of sampled data. It takes $|\mathcal{I}|$ samples of input at a time and takes the weighted average of those to produce a single output point. As the

length of the filter increases, the smoothness of the output increases, whereas the sharp modulations in the data are smoothed out.

The moving average filter is in fact a convolution using a very simple filter kernel. More generally, we can write a univariate time series prediction problem as a convolution with a filter as follows. First, the discrete convolution gives the relation between x_i and x_j :

$$x_{t-i} = \sum_{j=0}^{t-1} \delta_{ij} x_{t-j}, \quad i \in \{0, \dots, t-1\} \quad (8.58)$$

where we have used the Kronecker delta δ . The kernel filtered time series is a convolution

$$\tilde{x}_{t-i} = \sum_{j \in J} K_{j+k+1} x_{t-i-j}, \quad i \in \{k+1, \dots, p-k\}, \quad (8.59)$$

where $J := \{-k, \dots, k\}$ so that the span of the filter $|J| = 2k + 1$, where k is taken as a small integer, and the kernel is K . For simplicity, the ends of the sequence are assumed to be unfiltered but for notational reasons we set $\tilde{x}_{t-i} = x_{t-i}$ for $i \in \{1, \dots, k, p-k+1, \dots, p\}$. Then the filtered $AR(p)$ model is

$$\hat{x}_t = \mu + \sum_{i=1}^p \phi_i \tilde{x}_{t-i} \quad (8.60)$$

$$= \mu + (\phi_1 L + \phi_2 L^2 + \dots + \phi_p L^p)[\tilde{x}_t] \quad (8.61)$$

$$= \mu + [L, L^2, \dots, L^p] \boldsymbol{\phi}[\tilde{x}_t], \quad (8.62)$$

with coefficients $\boldsymbol{\phi} := [\phi_1, \dots, \phi_p]$. Note that there is no look-ahead bias because we do not filter the last k values of the observed data $\{x_s\}_{s=1}^t$. We have just written our first toy 1D CNN consisting of a feedforward output layer and a non-activated hidden layer with one unit (i.e., kernel):

$$\hat{x}_t = W_y z_t + b_y, \quad z_t = [\tilde{x}_{t-1}, \dots, \tilde{x}_{t-p}]^T, \quad W_y = \boldsymbol{\phi}^T, \quad b_y = \mu, \quad (8.63)$$

where \tilde{x}_{t-i} is the i th output from a convolution of the p length input sequence with a kernel consisting of $2k + 1$ weights. These weights are fixed over time and hence the CNN is only suited to prediction from stationary time series. Note also, in contrast to a RNN, that the size of the weight matrix W_y increases with the number of lags in the model.

The univariate CNN predictor with p lags and H activated hidden units (kernels) is

$$\hat{x}_t = W_y \text{vec}(z_t) + b_y \quad (8.64)$$

$$[z_t]_{i,m} = \sigma \left(\sum_{j \in J} K_{m,j+k+1} x_{t-i-j} + [b_h]_m \right) \quad (8.65)$$

$$= \sigma(K * x_t + b_h), \quad (8.66)$$

where $m \in \{1, \dots, H\}$ denotes the index of the kernel and the kernel matrix $K \in \mathbb{R}^{H \times 2k+1}$, hidden bias vector $b_h \in \mathbb{R}^H$ and output matrix $W_y \in \mathbb{R}^{1 \times pH}$.

Dimension Reduction

Since the size of W_y increases with both the number of lags and the number of kernels, it may be preferable to reduce the dimensionality of the weights with an additional layer and hence avoid over-fitting. We will return to this concept later, but one might view it as an alternative to auto-shrinkage or dropout.

Non-sequential Models

Convolutional neural networks are not limited to sequential models. One might, for example, sample the past lags non-uniformly so that $I = \{2^i\}_{i=1}^p$ then the maximum lag in the model is 2^p . Such a non-sequential model allows a large maximum lag without capturing all the intermediate lags. We will also return to non-sequential models in the section on dilated convolution.

Stationarity

A univariate CNN predictor, with one kernel and no activation, can be written in the canonical form

$$\hat{x}_t = \mu + (1 - \Phi(L))[K * x_t] = \mu + K * (1 - \Phi(L))[x_t] \quad (8.67)$$

$$= \mu + (\tilde{\phi}_1 L + \dots + \tilde{\phi}_p L^p)[x_t] \quad (8.68)$$

$$:= \mu + (1 - \tilde{\Phi}(L))[x_t], \quad (8.69)$$

where, by the linearity of $\Phi(L)$ in x_t , the convolution commutes and thus we can write $\tilde{\phi} := K * \phi$. Finding the roots of the characteristic equation

$$\tilde{\Phi}(z) = 0, \quad (8.70)$$

it follows that the CNN is strictly stationary and ergodic if all the roots lie outside the unit circle in the complex plane, $|\lambda_i| > 1$, $i \in \{1, \dots, p\}$. As before, we would compute the eigenvalues of the companion matrix to find the roots. Provided that $\tilde{\Phi}(L)^{-1}$ forms a divergent sequence in the noise process $\{\epsilon_s\}_{s=1}^t$ then the model is *stable*.

5.2 2D Convolution

2D convolution involves applying a small kernel matrix (a.k.a. a filter), $K \in \mathbb{R}^{2k+1 \times 2k+1}$, over the input matrix (called an image), $X \in \mathbb{R}^{m \times n}$, to give a filtered image, $Y \in \mathbb{R}^{m-2k \times n-2k}$. In the context of convolutional neural networks, the elements of the filtered image are referred to as the *feature map* values and are calculated according to the following formula:

$$y_{i,j} = [K * X]_{i,j} = \sum_{p,q=-k}^k K_{k+1+p,k+1+q} x_{i+p+1,j+q+1}, \\ i \in \{1, \dots, m\}, j \in \{1, \dots, n\}. \quad (8.71)$$

It is instructive to consider the following example to illustrate the 2D convolution with a small kernel matrix.

Example 8.2 2D Convolution

Consider the 4×4 input, 3×3 kernel, and 2×2 output matrices

$$X = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 3 \\ 2 & 0 & 1 & 0 \\ 0 & 2 & 1 & 0 \end{bmatrix}, \quad K = \begin{bmatrix} 0 & -1 & 1 \\ 0 & 1 & 0 \\ 1 & -1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 2 \\ -2 & 5 \end{bmatrix}. \quad \text{The calculation of the}$$

outputs for the case when $i = j = 1$ is

$$y_{i,j} = [K * X]_{i,j} = \sum_{p,q=-k}^k K_{k+1+p,k+1+q} x_{i+p+1,j+q+1}, \\ i \in \{1, \dots, m\}, j \in \{1, \dots, n\} \\ = 0 \cdot 1 + -1 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 2 + -1 \cdot 0 + 0 \cdot 1 \\ = 2$$

We leave it as an exercise for the reader to compute the output for the remaining values of i and j .

As in the example above, when we perform convolution over the 4×4 image with a 3×3 kernel, we get a 2×2 feature map. This is because there are only 4 unique positions where we can place our filter inside this image.

As convolutional neural networks were designed for image processing, it is common to represent the color values of the pixels with c color channels. For

example, RGB values are represented with three channels. The general form of the convolution layer map for a $m \times n \times c$ input tensor and outputs $m \times n \times H$ (with stride 1 and padding) is

$$\theta : \mathbb{R}^{m \times n \times c} \rightarrow \mathbb{R}^{m \times n \times H}.$$

Writing

$$f = \begin{pmatrix} f_1 \\ \vdots \\ f_c \end{pmatrix} \quad (8.72)$$

we can then write the layer map as

$$\theta(f) = K * f + \mathbf{b}, \quad (8.73)$$

where $K \in \mathbb{R}^{[(2k+1) \times (2k+1)] \times H \times c}$, $\mathbf{b} \in \mathbb{R}^{m \times n \times H}$ given by $\mathbf{b} := \mathbf{1}_{m \times n} \otimes b$ and $\mathbf{1}_{m \times n}$ is a $m \times n$ matrix with all elements being 1.

In component form, the operation (8.73) is

$$[\theta(f)]_j = \sum_{i=1}^c [K]_{i,j} * [f]_i + \mathbf{b}_j, \quad j \in \{1, \dots, H\}, \quad (8.74)$$

where $[.]_{i,j}$ contracts the 4-tensor to a 2-tensor by indexing the i th third component and j th fourth component of the tensor and for any $g \in \mathbb{R}^{m \times n}$ and $H \in \mathbb{R}^{(2k+1) \times (2k+1)}$

$$[H * g]_{i,j} = \sum_{p,q=-k}^k H_{k+1+p,k+1+q} g_{i+p,j+q}, \quad i \in \{1, \dots, m\}, j \in \{1, \dots, n\}. \quad (8.75)$$

By analogy to a fully connected feedforward architecture, the weights in the layer are given by the kernel tensor, K , and the biases, b are H -vectors. Instead of a semi-affine transformation, the layer is given by an activated convolution $\sigma(\theta(\mathbf{f}))$.

Furthermore, we note that not all neurons in the two consecutive layers are connected to each other. In fact, only the neurons which correspond to inputs within a $2k + 1 \times 2k + 1$ square connect to the same output neuron. Thus the filter size controls the *receptive field* of each output. We note, therefore, that some neurons share the same weights. Both of these properties result in far fewer parameters to learn than a fully connected feedforward architecture.

Padding is needed to extend the size of the image f so that the filtered image has the same dimensions as the original image. Specifically padding means how to choose $f_{i+p,j+q}$ when $(i + p, j + q)$ is outside of $\{1, \dots, m\}$ or $\{1, \dots, n\}$. The following three choices are often used

$$f_{i+p, j+q} = \begin{cases} 0, & \text{zero padding,} \\ f_{(i+p) \pmod m, (s+q) \pmod n}, & \text{periodic padding,} \\ f_{|i-1+p|, |j-1+q|}, & \text{reflected padding,} \end{cases} \quad (8.76)$$

if

$$i + p \notin \{1, \dots, m\} \text{ or } j + q \notin \{1, \dots, n\}. \quad (8.77)$$

Here $d \pmod m \in \{1, \dots, m\}$ means the remainder when d is divided by m .

The operation in Eq. 8.75 is also called a convolution with stride 1. Informally, we performed the convolution by sliding the image area by a unit increment. A common choice in CNNs is to take $s = 2$. Given an integer $s \geq 1$, a convolution with stride s for $f \in \mathbb{R}^{m \times n}$ is defined as

$$[K *_s f]_{i,j} = \sum_{p,q=-k}^k K_{p,q} f_{s(i-1)+1+p, s(j-1)+1+q}, \quad i \in \{1, \dots, \lceil \frac{m}{s} \rceil\}, \\ j \in \{1, \dots, \lceil \frac{n}{s} \rceil\}. \quad (8.78)$$

Here $\lceil \frac{m}{s} \rceil$ denotes the smallest integer greater than $\frac{m}{s}$.

5.3 Pooling

Data with high spatial structure often results in observations which have similar values within a neighborhood. Such a characteristic leads to redundancy in data representation and motivates the use of data reduction techniques such as pooling. In addition to a convolution layer, a pooling layer is a map:

$$\bar{R}_\ell^{\ell+1} : \mathbb{R}^{m_\ell \times n_\ell} \rightarrow \mathbb{R}^{m_{\ell+1} \times n_{\ell+1}}. \quad (8.79)$$

One popular pooling is the so-called average pooling R_{avr} which can be a convolution with stride 2 or bigger using the kernel K in the form of

$$K = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}. \quad (8.80)$$

Non-linear pooling operator is also used, for example, the $(2k + 1) \times (2k + 1)$ max-pooling operator with stride s as follows:

$$[R_{\max}(f)]_{i,j} = \max_{-k \leq p, q \leq k} \{f_{s(i-1)+1+p, s(j-1)+1+q}\}. \quad (8.81)$$

5.4 Dilated Convolution

In addition to image processing, CNNs have also been successfully applied to time series. WaveNet, for example, is a CNN developed for audio processing (van den Oord et al. 2016).

Time series often displays long-term correlations. Moreover, the dependent variable(s) may exhibit non-linear dependence on the lagged predictors. The WaveNet architecture is a non-linear p -autoregression of the form

$$y_t = \sum_{i=1}^p \phi_i(x_{t-i}) + \epsilon_t \quad (8.82)$$

where the coefficient functions $\phi_i, i \in \{1, \dots, p\}$ are data-dependent and optimized through the convolutional network.

To enable the network to learn these long-term, non-linear, dependencies Borovskykh et al. (2017) use stacked layers of dilated convolutions. A dilated convolution effectively allows the network to operate on a coarser scale than with a normal convolution. This is similar to pooling or strided convolutions, but here the output has the same size as the input (van den Oord et al. 2016).

In a dilated convolution the filter is applied to every d th element in the input vector, allowing the model to efficiently learn connections between far-apart data points. For an architecture with L layers of dilated convolutions $\ell \in \{1, \dots, L\}$, a dilated convolution outputs a stack of “feature maps” given by

$$[K^{(\ell)} *_{d^{(\ell)}} f^{(\ell-1)}]_i = \sum_{p=-k}^k K_p^{(\ell)} f_{d^{(\ell)}(i-1)+1+p}^{(\ell-1)}, \quad i \in \{1, \dots, \lceil \frac{m}{d^{(\ell)}} \rceil\}, \quad (8.83)$$

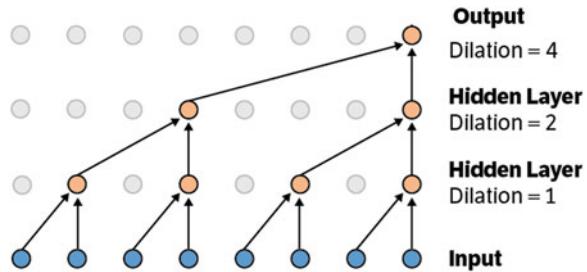
where d is the dilation factor and we can choose the dilations to increase by a factor of two: $d^{(\ell)} = 2^{\ell-1}$. The filters for each layer, $K^{(\ell)}$, are chosen to be of size $1 \times (2k+1) = 1 \times 2$.

An example of a three-layer dilated convolutional network is shown in Fig. 8.6. Using the dilated convolutions instead of regular ones allows the output y to be influenced by more nodes in the input. The input of the network is given by the time series X . In each subsequent layer we apply the dilated convolution, followed by a non-linearity, giving the output feature maps $f^{(\ell)}, \ell \in \{1, \dots, L\}$.

Since we are interested in forecasting the subsequent values of the time series, we will train the model so that this output is the forecasted time series $\hat{Y} = \{\hat{y}_t\}_{t=1}^N$.

The receptive field of a neuron was defined as the set of elements in its input that modifies the output value of that neuron. Now, we define the receptive field r of the model to be the number of neurons in the input in the first layer, i.e. the time series, that can modify the output in the final layer, i.e. the forecasted time series. This then depends on the number of layers L and the filter size $2k+1$, and is given by

Fig. 8.6 A dilated convolutional neural network with three layers. The receptive field is given by $r = 8$, i.e. one output value is influenced by eight input neurons. Source: van den Oord et al. (2016)



$$r := 2^{L-1}(2k+1). \quad (8.84)$$

In Fig. 8.6, the receptive field is given by $r = 8$, one output value is influenced by eight input neurons.

? Multiple Choice Question 3

Which of the following statements are true:

- a. CNNs apply a collection of different, but equal width, filters to the data before using a feedforward network for regression or classification.
 - b. CNNs are sparse networks, exploiting locality of the data, to reduce the number of weights.
 - c. A dilated CNN is appropriate for multi-scale time series analysis—it captures a hierarchy of patterns at different resolutions (i.e., dependencies on past lags at different frequencies, e.g. days, weeks, months)
 - d. The number of layers in a CNN is automatically determined during training.
-

5.5 Python Notebooks

`ML_in_Finance-1D-CNNs.ipynb` demonstrates the application of 1D CNNs to predict the next element in a uniform sequence of integers. The CNN uses a sequence length of 50 and 4 kernels each of width 5. See Exercise 8.7 for a programming challenge involving applying this 1D CNN for time series to the HFT dataset described in the previous section on RNNs.

For completeness, `ML_in_Finance-2D-CNNs.ipynb` demonstrates the application of a 2D CNN to image data from the MNIST dataset. Such an architecture might be appropriate for learning volatility surfaces but is not demonstrated here.

6 Autoencoders

An autoencoder is a *self-supervised* deep learner which trains the architecture to approximate the identity function, $Y = F(Y)$, via a bottleneck structure. This means we fit a model $\hat{Y} = F_{W,b}(Y)$ which aims to very efficiently concentrate the information required to recreate Y . Put differently, an autoencoder is a form of compression that creates a much more cost-effective representation of Y .

Its output layer has the same number of nodes as the input layer, and the cost function is some measure of the reconstruction error, $Y - \hat{Y}$. Autoencoders are often used for the purpose of dimensionality reduction and noise reduction. A simple autoencoder that implements dimensionality reduction is a feedforward autoencoder with at least one layer that has a smaller number of nodes, which functions as a bottleneck. After training the neural network using back-propagation, it is separated into two parts: the layers up to the bottleneck are used as an encoder, and the remaining layers are used as a decoder. In the simplest case, there is only one hidden layer (the bottleneck), and the layers in the network are fully connected. The compression capacity of autoencoders motivates their application in finance as a non-parametric, non-linear, analogue of the heavily used principal component analysis (PCA). It has been well known since the pioneering work of Baldi and Hornik (1989) that autoencoders are closely related to PCA. We follow Plaut (2018) and begin with a brief review of PCA and then show how exactly linear autoencoders enable PCA.

Example 8.3 A Simple Autoencoder

For example, under a L_2 -loss function, we wish to solve

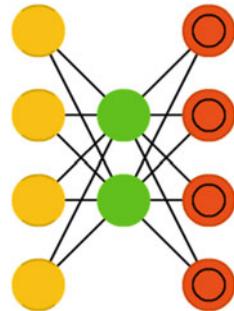
$$\underset{W,B}{\text{minimize}} \quad ||F_{W,b}(X) - Y||_F^2$$

subject to a regularization penalty on the weights and offsets. An autoencoder with two layers can be written as a feedforward network:

$$\begin{aligned} Z^{(1)} &= f^{(1)}(W^{(1)}Y + b^{(1)}), \\ \hat{Y} &= f^{(2)}(W^{(2)}Z^{(1)} + b^{(2)}), \end{aligned}$$

where $Z^{(1)}$ is a low-dimensional representation of Y . We find the weights and biases so that the number of rows of $W^{(1)}$ equals the number of columns of $W^{(2)}$ and the number of rows is much smaller than the columns, which looks like the architecture in Fig. 8.7.

Fig. 8.7 Autoencoders.
 Source: Van Veen, F. & Leijnen, S. (2019), “The Neural Network Zoo”, Retrieved from <https://www.asimovinstitute.org/neural-network-zoo>



6.1 Linear Autoencoders

In the case that no non-linear activation function is used, $\mathbf{x}_i = W^{(1)}\mathbf{y}_i + \mathbf{b}^{(1)}$ and $\hat{\mathbf{y}}_i = W^{(2)}\mathbf{x}_i + \mathbf{b}^{(2)}$. If the cost function is the total squared difference between output and input, then training the autoencoder on the input data matrix \mathbf{Y} solves

$$\min_{W^{(1)}, \mathbf{b}^{(1)}, W^{(2)}, \mathbf{b}^{(2)}} \left\| \mathbf{Y} - \left(W^{(2)} \left(W^{(1)} \mathbf{Y} + \mathbf{b}^{(1)} \mathbb{1}_N^T \right) + \mathbf{b}^{(2)} \mathbb{1}_N^T \right) \right\|_F^2. \quad (8.85)$$

If we set the partial derivative with respect to \mathbf{b}_2 to zero and insert the solution into (8.85), then the problem becomes

$$\min_{W^{(1)}, W^{(2)}} \left\| \mathbf{Y}_0 - W^{(2)} W^{(1)} \mathbf{Y}_0 \right\|_F^2$$

Thus, for any \mathbf{b}_1 , the optimal \mathbf{b}_2 is such that the problem becomes independent of \mathbf{b}_1 and of $\bar{\mathbf{y}}$. Therefore, we may focus only on the weights $W^{(1)}$, $W^{(2)}$.

Linear autoencoders give orthogonal projections, even though the columns of the weight matrices are not orthogonal. To see this, set the gradients to zero, $W^{(1)}$ is the left Moore–Penrose pseudoinverse of $W^{(2)}$ (and $W^{(2)}$ is the right pseudoinverse of $W^{(1)}$):

$$W^{(1)} = (W^{(2)})^\dagger = \left(W^{(2)T} W^{(2)} \right)^{-1} (W^{(2)})^T$$

The minimization with respect to a single matrix is

$$\min_{W^{(2)} \in \mathbb{R}^{n \times m}} \left\| \mathbf{Y}_0 - W^{(2)} (W^{(2)})^\dagger \mathbf{Y}_0 \right\|_F^2 \quad (8.86)$$

The matrix $W^{(2)} (W^{(2)})^\dagger = W^{(2)} \left((W^{(2)})^T W^{(2)} \right)^{-1} (W^{(2)})^T$ is the orthogonal projection operator onto the column space of $W^{(2)}$ when its columns are not

necessarily orthonormal. This problem is very similar to (6.52), but *without* the orthonormality constraint.

It can be shown that $W^{(2)}$ is a minimizer of Eq. 8.86 if and only if its column space is spanned by the first m loading vectors of Y .

The linear autoencoder is said to apply PCA to the input data in the sense that its output is a projection of the data onto the low-dimensional principal subspace. However, unlike actual PCA, the coordinates of the output of the bottleneck are *correlated* and are *not sorted in descending order of variance*. The solutions for reduction to different dimensions are not nested: when reducing the data from dimension n to dimension m_1 , the first m_2 vectors ($m_2 < m_1$) are not an optimal solution to reduction from dimension n to m_2 , which therefore requires training an entirely new autoencoder.

6.2 Equivalence of Linear Autoencoders and PCA

Theorem *The first m loading vectors of Y are the first m left singular vectors of the matrix $W^{(2)}$ which minimizes (8.86).* \square

A sketch of the proof now follows. We train the linear autoencoder on the original dataset Y and then compute the first m left singular vectors of $W^{(2)} \in \mathbb{R}^{n \times m}$, where typically $m \ll N$. The loading vectors may also be recovered from the weights of the hidden layer, $W^{(1)}$, by a singular value decomposition. If $W^{(2)} = U\Sigma V^T$, which we assume is full-rank, then

$$W^{(1)} = (W^{(2)})^\dagger = V\Sigma^\dagger U^T$$

and

$$W_2 W_2^\dagger = U\Sigma V^T V\Sigma^\dagger U^T = U\Sigma\Sigma^\dagger U^T = U_m U_m^T, \quad (8.87)$$

where we used the fact that $(V^T)^\dagger = V$ and that $\Sigma^\dagger \in \mathbb{R}^{m \times n}$ is a matrix whose diagonal elements are $\frac{1}{\sigma_j}$ (assuming $\sigma_j \neq 0$, and 0 otherwise).

The matrix $\Sigma\Sigma^\dagger$ is a diagonal matrix whose first m diagonal elements are equal to one and the other $n - m$ elements are equal to zero. The matrix $U_m \in \mathbb{R}^{n \times m}$ is a matrix whose columns are the first m left singular vectors of W_2 . Thus, the first m left singular vectors of $(W^{(1)})^T \in \mathbb{R}^{n \times m}$ are also equal to the first m loading vectors of Y .

A common application of PCA is in fixed income modeling—the principal components are used to characterize the daily movement of the yield curve. Because the components explain most of the variability in the curve, investors can hedge their exposures with only a few instruments from different sectors (Litterman and

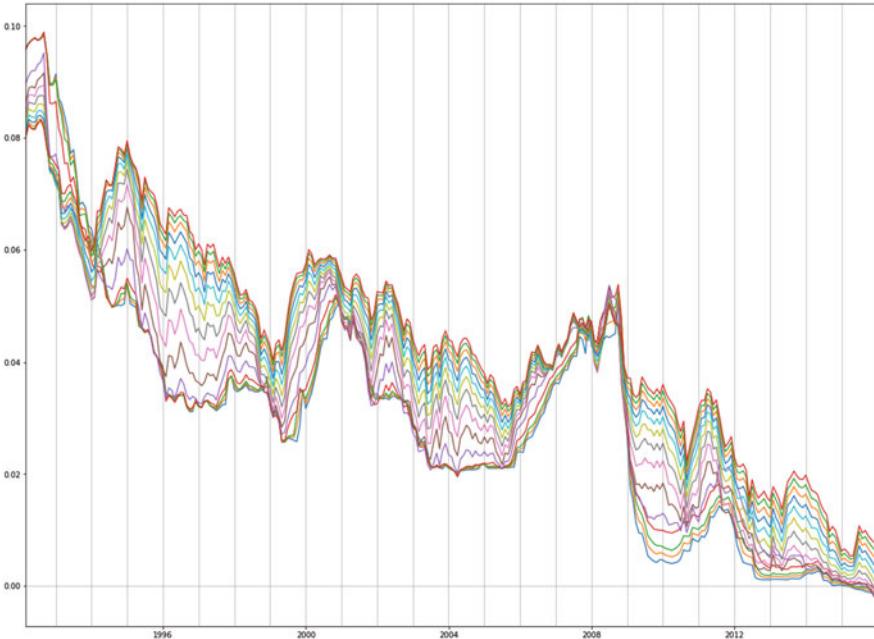


Fig. 8.8 This figure shows the yield curve over time, each line corresponds to a different maturity in the term structure of interest rates

Scheinkman 1991). Figure 8.8 shows the yield curve over a 25-year period, where each line corresponds to each of the maturities in the term structure of fixed income securities.

We can illustrate the comparison by finding the principal components of the sample covariance matrix from the time series of the yield curve, as shown in Fig. 8.9a. The eigenvalues are the diagonal of the transformed matrix, are all positive, and arranged in descending order. In this case we have plotted the first $m = 3$ components from a high-dimensional dataset where $n > m$. The percentage of variance attributed to these components is 95.6%, 4.07%, 0.34%, respectively. Figure 8.9c shows the decomposition of the sample covariance matrix using the left singular vectors of the autoencoder weights and observed to be similar to Fig. 8.9a. The percentage of variance attributed to the components is 95.63%, 4.10%, 0.27%. For completeness, Fig. 8.9b shows the transformation using $W^{(2)}$, which results in correlated values.

Performing PCA on the daily change of the yield curve leads to more interpretable components: the first eigenvalue can be attributed to parallel shift of the curve, the second to twist, and the third to curvature (a.k.a. butterfly). Figure 8.10 compares the first two principal components of ΔY_0 using either the $m = 3$ loading vectors, P_m or the $m = 3$ singular vectors, U_m . For the purposes of interpreting the behavior of the yield curve over time, both give similar results. Periods in which

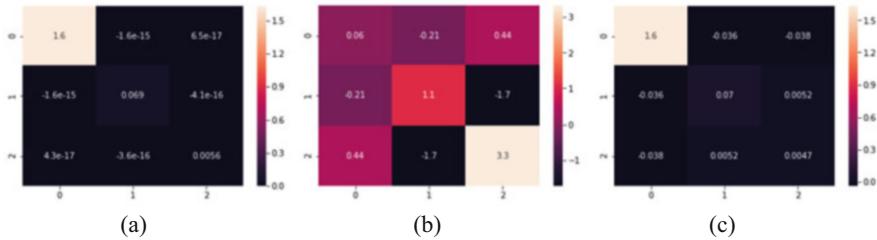


Fig. 8.9 The covariance matrix of the data in the transformed coordinates, according to (a) the loading vectors computed by applying SVD to the entire dataset, (b) the weights of the linear autoencoder, and (c) the left singular vectors of the autoencoder weights. (a) $P_m^T Y_0 Y_0^T P_m$, (b) $(W^{(2)})^T Y_0 Y_0^T W^{(2)}$, (c) $U_m^T Y_0 Y_0^T U_m$

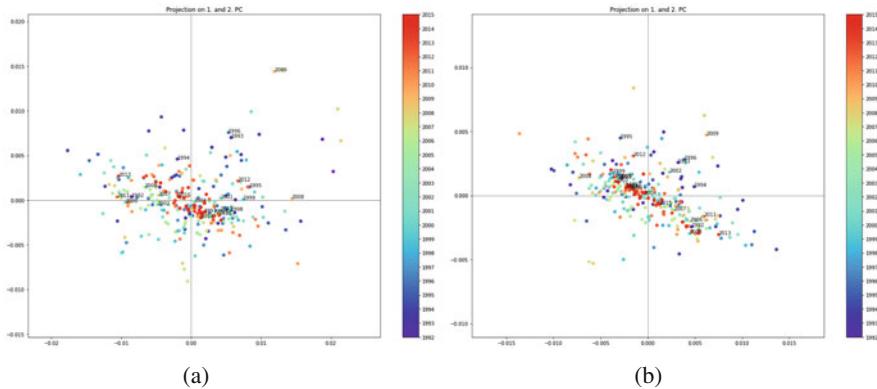


Fig. 8.10 The first two principal components of ΔY_0 , projected using P_m are shown in (a). The first two approximated principal components (up to a sign change) using U_m . The first principal component is represented by the x-axis and the second by the y-axis. (a) $P_m^T \Delta Y_0$. (b) $U_m^T \Delta Y_0$

the yield curve is dominated by parallel shift exhibit a large absolute first principal component compared to the second component. And conversely, periods exhibiting a large second component compared to the first indicates that the curve movement is dominated by twist. The latter phenomenal often occurs when the curve moves from an upward sloping to a download sloping regime. In both cases, we note that the period following the financial crisis, 2009, exhibits a relatively large amount of shift and twist when compared to other years.

6.3 Deep Autoencoders

As we saw in the previous chapter, merely adding more layers to the linear autoencoder does not change the properties of the autoencoder—it remains a linear

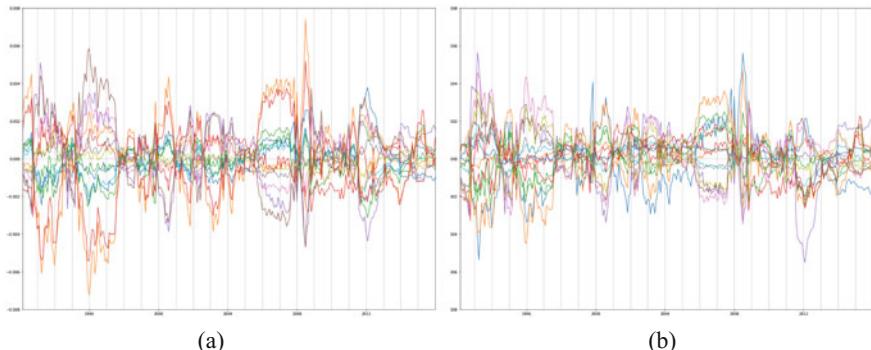


Fig. 8.11 The reconstruction error in Y is shown for **(a)** the linear encoder and **(b)** a deep autoencoder, with two \tanh activated layers for each of the encoder and decoders

autoencoder and if there are L layers in the encoder, then the first m singular values of $W^{(1)} W^{(2)} \dots W^{(L)}$ will correspond to the loading vectors. With non-linear activation, the autoencoder can no longer resolve the loading vectors. However, the addition of a more expressive, non-linear, model is used to reduce the reconstruction error for a given compression dimension m . Figure 8.11 compares the reconstruction error in Y using the linear autoencoder and a deep autoencoder, with two \tanh activated layers in each of the encoder and decoder.

Recently, the application of deep autoencoders to statistical equity factor models has been demonstrated by Heaton et al. (2017). The authors compress the asset returns in a portfolio to give a small set of deep factors which explain the variability in the portfolio returns more reliably than PCA or fundamental equity factors. Such a representation provides a general portfolio selection process which relies on encoding of the asset return histories into deep factors and then decoding, to predict the asset returns. One practical challenge with this approach, and indeed all statistical factor models, is their lack of investability and hedgeability. For ReLU activated autoencoders, deep factors can be interpreted as compositions of financial put and call options on linear combinations of the assets represented. As such, the authors speculate that deep factors could be potentially investable and hence hedgeable.

See the notebook `ML_in_Finance-Autoencoders.ipynb` for an implementation of the methodology and results presented in this section.

7 Summary

In this chapter we have seen how different neural network architectures can be used to exploit the structure in the data, resulting in fewer weights, and broadening their application as a wider class of models than regression and classification.

- We characterize RNNs as non-linear autoregressive models with geometrically decaying lagged coefficients. RNNs can be shown to exhibit unconditional stability under certain constraints on the activation function. In particular, a tanh activation will lead to a stable architecture;
- We combine hypothesis tests from classical time series analysis to guide the experimental design and diagnose the RNN output. In particular, we can sample the partial autocorrelation to determine the sequence length if the data is stationary and we can check for autocorrelation in the model error to determine if the model has under-fitted.
- Gated recurrent units and long short-term memory architectures give a dynamic autoregressive model with variable memory. Adaptive exponential smoothing is used to propagate a hidden variable with potentially infinite memory. These architectures have the ability to behave as plain RNNs or even as feedforward architectures, i.e. they behave as static non-linear autoregressive models or linear regression as a special case.
- CNNs filter the data and then exploit data locality, either spatial, temporal, or even spatio-temporal, to efficiently represent the input data. When applied to time series, CNNs are non-linear autoregressive models which can be designed to capture multiple scales in the data using dilated convolution.
- Principal component analysis is one of the most powerful techniques for dimension reduction and uses orthogonal projection to decorrelate the features.
- The first m singular values of the weight matrix in a linear autoencoder are the m loading vectors used as an orthogonal basis for projection.
- We can combine these different architectures together to build powerful regressions and compression methods. For example, we might use a GRU-autoencoder to compress non-stationary time series where as we might use a CNN autoencoder to compress spatial data.

8 Exercises

Exercise 8.1*

Calculate the half-life of the following univariate RNN

$$\begin{aligned}\hat{x}_t &= W_y z_{t-1} + b_y, \\ z_{t-1} &= \tanh(W_z z_{t-2} + W_x x_{t-1}),\end{aligned}$$

where $W_y = 1$, $W_z = W_x = 0.5$, $b_h = 0.1$ and $b_y = 0$.

Question 8.2: Recurrent Neural Networks

- State the assumptions needed to apply plain recurrent neural networks to time series data.
- Show that a linear RNN(p) model with bias terms in both the output layer and the hidden layer can be written in the form

$$\hat{y}_t = \mu + \sum_{i=1}^p \phi_i y_{t-i}$$

and state the form of the coefficients $\{\phi_i\}$.

- State the conditions on the activation function and weights in a plain RNN under which the model is stable? (i.e., lags do not grow)

Exercise 8.3*

Using Jensen's inequality, calculate the lower bound on the partial autocovariance function of the following zero-mean RNN(1) process:

$$y_t = \sigma(\phi y_{t-1}) + u_t,$$

for some monotonically increasing, positive and convex activation function, $\sigma(x)$ and positive constant ϕ . Note that Jensen's inequality states that $\mathbb{E}[g(X)] \geq g(\mathbb{E}[X])$ for any convex function g of a random variable X .

Exercise 8.4*

Show that the discrete convolution of the input sequence $X = \{3, 1, 2\}$ and the filter $F = \{3, 2, 1\}$ given by $Y = X * F$ where

$$y_i = X * F_i = \sum_{j=-\infty}^{\infty} x_j F_{i-j}$$

is $Y = \{9, 9, 11, 5, 2\}$.

Exercise 8.5*

Show that the discrete convolution $\hat{x}_t = F * x_t$ defines a univariate $AR(p)$ if a p -width filter is defined as $F_j := \phi^j$ for some constant parameter ϕ .

8.1 Programming Related Questions*

Exercise 8.6***

Modify the RNN notebook to predict Coindesk prices using a univariate RNN applied to the data `coindesk.csv`. Then complete the following tasks

- a. Determine whether the data is stationary by applying the augmented Dickey–Fuller test.
- b. Estimate the partial autocorrelation and determine the optimum lag at the 99% confidence level. Note that you will not be able to draw conclusions if your data is not stationary. Choose the sequence length to be equal to this optimum lag.
- c. Evaluate the MSE in-sample and out-of-sample as you vary the number of hidden neurons. What do you conclude about the level of over-fitting?

- d. Apply L_1 regularization to reduce the variance.
- e. How does the out-of-sample performance of a plain RNN compare with that of a GRU?
- f. Determine whether the model error is white noise or is auto-correlated by applying the Ljung–Box test.

Exercise 8.7***

Modify the CNN 1D time series notebook to predict high-frequency mid-prices with a single hidden layer CNN, using the data `HFT.csv`. Then complete the following tasks

- a. Confirm that the data is stationary by applying the augmented Dickey–Fuller test.
- b. Estimate the partial autocorrelation and determine the optimum lag at the 99% confidence level.
- c. Evaluate the MSE in-sample and out-of-sample using 4 filters. What do you conclude about the level of over-fitting as you vary the number of filters?
- d. Apply L_1 regularization to reduce the variance.
- e. Determine whether the model error is white noise or is auto-correlated by applying the Ljung–Box test.

Hint: You should also review the HFT RNN notebook before you begin this exercise.

Appendix

Answers to Multiple choice questions

Question 1

Answer: 1,2,4,5. An augmented Dickey–Fuller test can be applied to time series to determine whether they are covariance stationary.

The estimated partial autocorrelation of a covariance stationary time series can be used to identify the design sequence length in a plain RNN because the network has a fixed partial autocorrelation matrix.

Plain recurrent neural networks are not guaranteed to be stable—the stability constraint restricts the choice of activation in the hidden state update.

Once the model is fitted, the Ljung–Box test is used to test whether the residual error is auto-correlated. A well-specified model should exhibit white noise error both in and out-of-sample.

The half-life of a lag-1 unit impulse is the number of lags before the impulse has half its effect on the model output.

Question 2

Answer: 1,4. A gated recurrent unit uses dynamic exponential smoothing to propagate a hidden state with infinite memory. However, there is no requirement for

covariance stationarity of the data in order to fit a GRU, or LSTM. This is because the later are dynamic models with a time-dependent partial autocorrelation structure.

Gated recurrent units are conditionally stable—the choice of activation in the hidden state update is especially important. For example, a tanh function for the hidden state update satisfies the stability constraint. A GRU only has one memory, the hidden state, whereas a LSTM indeed has an additional, cellular, memory.

Question 3

Answer: 1,2,3.

CNNs apply a collection of different, but equal width, filters to the data. Each filter is a unit in the CNN hidden layer and is activated before using a feedforward network for regression or classification. CNNs are sparse networks, exploiting locality of the data, to reduce the number of weights. CNNs are especially relevant for spatial, temporal, or even spatio-temporal datasets (e.g., implied volatility surfaces). A dilated CNN, such as the WaveNet architecture, is appropriate for multi-scale time series analysis—it captures a hierarchy of patterns at different resolutions (i.e., dependencies on past lags at different frequencies, e.g., days, weeks, months). The number of layers in a CNN must be determined manually during training.

Python Notebooks

The notebooks provided in the accompanying source code repository implement many of the techniques presented in this chapter including RNNs, GRUs, LSTMs, CNNs, and autoencoders. Example datasets include 1-minute snapshots of Coinbase prices and a HFT dataset. Further details of the notebooks are included in the README .md file.

References

- Baldi, P., & Hornik, K. (1989, January). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Netw.*, 2(1), 53–58.
- Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017, Mar). Conditional time series forecasting with convolutional neural networks. *arXiv e-prints*, arXiv:1703.04691.
- Elman, J. L. (1991, Sep). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2), 195–225.
- Gers, F. A., Eck, D., & Schmidhuber, J. (2001). *Applying LSTM to time series predictable through time-window approaches* (pp. 669–676). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Graves, A. (2012). *Supervised sequence labelling with recurrent neural networks*. Studies in Computational intelligence. Heidelberg, New York: Springer.
- Heaton, J. B., Polson, N. G., & Witte, J. H. (2017). Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1), 3–12.
- Hochreiter, S., & Schmidhuber, J. (1997, November). Long short-term memory. *Neural Comput.*, 9(8), 1735–1780.

- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).
- Litterman, R. B., & Scheinkman, J. (1991). Common factors affecting bond returns. *The Journal of Fixed Income*, 1(1), 54–61.
- Plaut, E. (2018, Apr). From principal subspaces to principal components with linear autoencoders. *arXiv e-prints*, arXiv:1804.10253.
- van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., et al. (2016). WaveNet: A generative model for raw audio. *CoRR*, abs/1609.03499.
- Zheng, J., Xu, C., Zhang, Z., & Li, X. (2017, March). Electric load forecasting in smart grids using long-short-term-memory based recurrent neural network. In *2017 51st Annual Conference on Information Sciences and Systems (CISS)* (pp. 1–6).

Part III

Sequential Data with Decision-Making

Chapter 9

Introduction to Reinforcement Learning



This chapter introduces Markov Decision Processes and the classical methods of dynamic programming, before building familiarity with the ideas of reinforcement learning and other approximate methods for solving MDPs. After describing Bellman optimality and iterative value and policy updates before moving to Q-learning, the chapter quickly advances towards a more engineering style exposition of the topic, covering key computational concepts such as greediness, batch learning, and Q-learning. Through a number of mini-case studies, the chapter provides insight into how RL is applied to optimization problems in asset management and trading.

1 Introduction

In the previous chapters, we dealt with supervised and unsupervised learning. Recall that supervised learning involves training an agent to produce an output given an input, where a teacher provides some training examples of input–output pairs. The task of the agent is to generalize from these examples, that is to find a function that produces outputs given inputs which are consistent with examples provided by the teacher. In unsupervised learning, the task is again to generalize, i.e. provide some outputs given inputs; however, there is no teacher to provide examples of a “ground truth.”

In this chapter, we address a different type of learning where an agent should learn to *act* optimally, given a certain goal, in a setting of sequential decision-making given a state of its environment. The latter serves as an input, while the agent’s actions are outputs. Acting optimally given a goal is mathematically formulated as a problem of maximization of a certain objective function. Problems of such sort belong to the area of machine learning known as of reinforcement learning (RL). Such an area of machine learning is tremendously important in trading and investment management.

Chapter Objectives

By the end of this chapter, the reader should expect to accomplish the following:

- Gain familiarity with Markov Decision Processes;
- Understand the Bellman equation and classical methods of dynamic programming;
- Gain familiarity with the ideas of reinforcement learning and other approximate methods of solving MDPs;
- Understand the difference between off-policy and on-policy learning algorithms; and
- Gain insight into how RL is applied to optimization problems in asset management and trading.

The notebooks provided in the accompanying source code repository accompany many of the examples in this chapter. See Appendix “Python Notebooks” for further details.

The task of finding an optimal mapping of inputs into outputs given an objective function looks superfluously similar to tasks of both supervised and unsupervised learning. Indeed, in all these cases, and in a sense in all problems of machine learning in general, the objective is always formulated as a sample-based problem of mapping some inputs into some outputs given some criteria for optimality of such mapping. This can be generally viewed as a special case of optimization or alternatively as a special case of function approximation. There are however at least three distinct differences between the problem setting in RL and the settings of both supervised learning and unsupervised learning.

The first difference is the presence and role of a teacher. In RL, like supervised learning and unlike unsupervised learning, there is a teacher. However, a feedback provided by the teacher to an agent is different from a feedback from a teacher in supervised learning. In the latter case, a teacher gives correct outputs for a given training dataset. The role of a supervised learning algorithm is to generalize from such explicit examples, i.e. to provide a function that maps any inputs to outputs, including inputs not encountered in the training set.

In reinforcement learning , a teacher provides only a *partial feedback* for actions taken by an agent. Such a partial feedback is given in terms of *rewards* that the agent receives upon taking a certain action. Rewards have numerical values, therefore a higher reward for a particular action generally implies that this particular action taken by the agent is better than other actions that would produce lower rewards. However, there is no *explicit* information from a teacher on what action is the best or is “right” to produce the highest reward possible. Therefore, a teacher in this case provides only a partial feedback to an agent during training. The goal of a RL agent is to maximize a total cumulative reward over a sequence of steps.

This brings us to a second key difference of reinforcement learning from supervised and unsupervised learning, which stems from a presence of a *feedback loop* from actions of the agent to a state of environment. This means that when the agent acts in a certain state of the environment, the action of the agent can change the state of the environment. Because reinforcement learning tasks involve sequential decision-making in an environment that is typically evolving stochastically on its own, as well as can be impacted by agent’s action via a feedback loop, reinforcement learning usually involves *planning*.

The presence of a feedback loop and the need for planning are unique to reinforcement learning. No feedback loop ever appears in supervised or unsupervised learning. Indeed, consider, for example, binary classification, a classical problem of supervised learning introduced in Chap. 1 and covered in more depth in Chap. 8. Assigning an output label to any particular input data can be viewed as an “action” of an agent in this setting. A classical zero-one loss function can be viewed as a negative reward. A distribution of true output labels in the remaining data points after a next data point is classified can be viewed as a state of an environment. Clearly, any particular choice of a label for a given data point in a test sample does not change true labels on the remaining data points. In other words, there is no feedback loop in such setting—the environment does not change as a result of “actions” of the agent. This is an important differentiator in finance when the actions change the environment, such as trades changing the state of the order book or consumption from a fund which changes the fund’s wealth.

Finally, a third difference between reinforcement learning and other types of learning is related to the previous two. Because the maximum attainable reward is not known to the agent, even when it achieves a high reward from performing a given action in a given state, there is always a possibility that taking some other action in the same state would produce yet a higher reward. The agent is therefore faced with a task of *exploration*, i.e. randomly selecting some other actions, in its quest for maximizing the total cumulative reward. On the other hand, by *not* reselecting actions that previously were found to produce high rewards, the agent can miss an opportunity for *exploitation*. Exploitation amounts to simply repeating actions producing “good” rewards, instead of taking the risk of producing smaller rewards as a result of exploration.

The agent is therefore faced with the so-called *exploration-exploitation dilemma* that conceptualizes the problem of choosing between exploration and exploitation at each step of a learning process. Additional complexity is introduced by the presence of a feedback loop. A particular action taken in a given state may produce a high local reward, but it may also change the environment and change future rewards as a result. Therefore, in choosing at each time step between exploration and exploitation, the agent should also do forecasting of future states that may be impacted by an agent’s actions.

The exploration-exploitation dilemma is specific to reinforcement learning and does not arise in supervised or unsupervised learning, because neither a partial feedback via rewards nor a feedback loop is present in these settings.

However, it is important to note that the exploration-exploitation dilemma is only applicable in a regime of “real-time” or *on-line* reinforcement learning. In this setting, an agent has access to a physical or simulated environment, and is therefore free to choose various actions and explore consequences, i.e. to engage in exploration.

Another situation arises in *batch-mode*, or off-line reinforcement learning. In this case, an agent does not have an on-demand access to an environment. Instead, it only has access to a dataset that stores a history of interactions of some other agents (human or machine) with this environment. These data should contain records of states of the environment, actions taken, and rewards received, for each time step in the history. In such a setting, the agent cannot explore, therefore the exploration-exploitation dilemma does not arise in this case. Batch-mode reinforcement learning is essentially a problem of inference of best possible actions given batch data of a recorded sequence of states, actions, and rewards received.

Batch-model reinforcement learning is important for finance, as its setting is similar to a traditional setting of financial models that are usually trained off-line, using some historical data. Our examples in the following chapters will mostly consider batch reinforcement learning, rather than its online version. On the other hand, batch and online reinforcement learning can be combined in certain cases. For example, an agent can be first trained off-line using batch reinforcement learning, and then be exposed to a real-time environment, where it can continue training online. Alternatively, online learning can be implemented as a version of batch learning using the so-called experience replay method, which amounts to adding each new combination of a state, action, and reward to a dataset of previously recorded such combinations, and gradually removing old observations so that the size of the experience replay buffer stays the same.

➤ Thompson Sampling

If interactions of an agent with its environment do *not* involve a feedback loop effect from the agent’s action on the evolution of the environment, a multi-step optimization of a total expected reward becomes equivalent to a sequence of independent one-step episodes, where states s_t of the environment provide a “context” for a decision policy $\pi(a|s = s_t)$. If a set of possible actions $a \in \mathcal{A}$ is discrete of size K , the problem is equivalent to a contextual Multi-Armed Bandit (MAB).

In applications such as online advertising or cloud design, arms of a bandit correspond to different features selected in order to optimize a certain score, e.g. a conversion rate of online users. In the setting of MABs as one-step reinforcement learning without a feedback loop, different arms correspond to different possible actions given the context s_t , while the objective of maximization of a total reward over a certain time horizon. In the financial

(continued)

context, a MAB setting may be well suited to describe long-term investors who choose between different stationary “all weather” strategies¹ to maximize a cumulative risk-adjusted reward.

Thompson sampling, also known as *posterior sampling* or probability matching, was first proposed by Thompson (1993, 1935) for two-armed bandit problems arising in clinical trials. Later it was established that the Thompson policy provides an optimal, in a certain sense, way to combine exploration and exploitation and manage their tradeoff for MAB problems. Thompson sampling (TS) is a simple Bayesian method which, at each time step, samples a list of arm means from the posterior distribution of optimal action, and chooses the best arm according to this sample. After that, the agent updates the posterior distribution using a Bayesian update with a reward observed as a result of picking the arm. In the financial setting, TS might serve as a useful technique to provide an optimal balance of exploration in exploitation in intraday trading, where arms would correspond to different trading strategies, or different specifications of the same trading strategy.

In more details, let $\mathbf{r}_t = (r_1, \dots, r_t)$ be a sequence of rewards obtained up to time t , where all rewards are considered for simplicity binary $r_t = \{0, 1\}$. We can consider a binomial bandit with the probability of success $f_\theta(y_t = 1|s_t) = g(\theta^T \mathbf{s}_t)$ where $g(\cdot)$ is a link function bounded to the interval $[0, 1]$, for example, it can be taken as a sigmoid function $g(z) = \sigma(z) = 1/(1+\exp(-z))$, and θ is a vector of model parameters. We denote $a_k \in \{1, \dots, K\}$ be the integer-valued arm (action) given the state s_t .

Let $\mu_a(\theta) = \mathbb{E}[y_t|\theta, a_t]$ be the expected reward from the distribution $f_a(y|\theta)$. If θ were known, then the optimal long-term strategy would be to always pick the arm with the highest value of $\mu_a(\theta)$. Thompson sampling uses a prior distribution on the parameter vector $p(\theta)$. After sampling of value of θ from this distribution, the best arm is decided according to the highest value of $\mu_a(\theta)$ across different arms. When an arm with a highest expected reward is executed, the posterior distribution of θ at time t becomes

$$p(\theta|\mathbf{y}_t) \sim p(\theta) \prod_{\tau=1}^t f_{a_\tau}(y_\tau|\theta). \quad (9.1)$$

The posterior probability $p(\theta|\mathbf{y}_t)$ is then used to update the best arm according to the highest expected reward criterion, and the execution continues to the next time step.

¹These are portfolio strategies that are not adaptive to changing macro-economical conditions and thus do not need to be rebalanced frequently.

2 Elements of Reinforcement Learning

Now that we have discussed the exploration-exploitation dilemma as a seminal concept in experimental design, we turn to more practical challenges of modeling with RL. The four main elements of reinforcement learning are (i) rewards; (ii) a value function; (iii) a policy; and (iv) an environment. In this section, we provide a high-level overview of these concepts.

2.1 Rewards

A reward function determines the *goal* of a reinforcement learning problem. The goal of an agent is to maximize the total (or average) reward received over a certain period of time. The reward received by an agent upon performing a certain action in a certain state can be thought as a measure of “happiness” from taking this action. For biological agents such as animals or humans, rewards can be considered numerical expressions of the experiences of pleasure or pain. The latter experiences could be represented as a set of rank-ordered (and possibly continuous) values that can be assigned to a specified interval.

The local (one-step) reward received by the agent in any given state of the environment is what determines the agent’s next action. Mathematically, we can write it as $r_t = r_t(S_t, a_t)$, i.e. a function that in general can depend on the current state S_t of the environment, and the action a_t taken in this state. In addition, for time-dependent problems, the reward can explicitly depend on time t , as implied by the index t in the expression for r_t . For time-independent problems, the time index can be omitted. In addition, the reward can have a random component that does not depend on the state S_t or action a_t . In this case, we deal with *random rewards*.

Sometimes, but not always, the best action that the agent can take is simply to choose an action that maximizes the local reward. This is a standard optimization problem that can be solved either exactly (if, e.g., the reward is a quadratic polynomial in action a_t), or numerically, for more complex forms of the reward function. This produces significant simplifications. For more insight, let us assume that the goal of an agent is to maximize the total reward obtained over T time steps. If rewards obtained at different steps are independent of each other, this problem is easy to solve: at every time step and in any state, the agent should simply pick the action that maximizes its local reward.

However, many cases of practical interest do not fit into this scheme. Complications arise due to the presence of a feedback loop that was mentioned above. Actions of an agent can change the environment. One simple example in finance is provided by the presence of market impact effect: if you trade a substantial amount of stocks or other asset, you move market prices by your action.

Because, in this case, actions of the agent can potentially change the evolution of the environment, rewards obtained by the agent at different time steps are no longer independent. Respectively, the problem of maximization of the total (cumulative) reward over T steps does not decompose into a set of T independent optimization problems. The problem then acquires a time dimension and becomes substantially more complex, calling for dedicated optimization methods. As we will see below, dynamic programming and reinforcement learning, as its extension, provide computationally efficient methods for “learning by acting” with a feedback loop. To illustrate the challenge of time-dependent optimization with actions which change the environment, let us consider the following optimal stock execution problem.

Example 9.1 Optimal Stock Execution

Optimal stock execution is a common problem that has to be solved thousands of times a day by large broker-dealers or trading desks of larger money managers. Assume a portfolio manager wants to sell a certain (large) quantity V of stocks of AMZN on NASDAQ within the next $T = 10$ min, and the objective is to maximize the total payoff. However, the latter is uncertain, due to both market fluctuations and a potential impact of trades to be executed following this order on the market price of the stock. If the broker who executes the order simultaneously sells $a_t \gg 1$ shares, this may drive the market price of the company down, so that executing the remaining shares will proceed at a smaller price, at a loss to the seller. A simple way to model market stock price dynamics with market impact is to use a linear impact model:

$$S_{t+1} = S_t(1 - \mu a_t) + \sigma Z_t, \quad (9.2)$$

where S_t and S_{t+1} are stock prices at times t and $t + \Delta t$, respectively, $a_t > 0$ is the trading volume of the sold stock, $Z_t \sim N(0, 1)$ is a standard Gaussian noise, and σ is the stock volatility. The problem faced by the broker is to find an optimal partitioning of the sell order into smaller blocks of shares that are to be executed sequentially, for example, each minute over the $T = 10$ min time period. This can also be formulated as a Markov Decision Process (MDP) problem (see Sect. 3 for further details) with a state variable X_t given by the number of shares outstanding (plus potentially other relevant variables such as limit order book data). For a one-step reward r_t of selling a_t shares at step t , we could consider a risk-adjusted payoff given by the total expected payoff $\mu := a_t S_t$ penalized by the variance of the remaining inventory price at the next step $t + 1$: $r_t = \mu - \lambda \text{Var}[S_{t+1} X_{t+1}]$. We will return to the optimal stock execution problem later in this chapter, after we introduce the relevant mathematical constructs.

2.2 Value and Policy Functions

In addition to an action chosen by the agent, its local reward depends on the state S_t of the environment. Different states of the environment can have different amount of attractiveness, or value, for the agent. Certain states may not exhibit any good options to receive high rewards. Furthermore, states of the environment in general change over a multi-step sequence of actions taken by the agent, and their future may be partly driven by their present state (and possibly actions of the agent). Therefore, reinforcement learning uses the concept of a *value function* as a numerical value of attractiveness of a state S_t for the agent, with a view on a multi-step character of an optimization problem faced by the agent.

To relate it to the goal of reinforcement learning of maximization the cumulative reward over a period of time, a value function can be specified as a mean (expected) cumulative reward, that can be obtained by starting from this state, over the whole period (but as we will see below, there are other choices too). However, such a quantity would be under-specified as it stands, because rewards depend on actions a_t , in addition to their dependence on the state S_t . If we want to define the value function of a current (time- t) expected value of the cumulative reward obtained in T steps, we must know *beforehand* how the agent should act in *any* given possible state of the environment.

This rule is specified by a *policy function* $\pi_t(S_t)$ for how the agent should act at time t given the state S_t of the environment. A policy function may be a deterministic function of the state S_t , or alternatively it can be a probability distribution over a range of possible actions, specified by the current value of S_t . A value function $V^\pi(S_t)$ is therefore a *function* of the current state S_t and a *functional* of the policy π .

2.3 Observable Versus Partially Observable Environments

Finally, to complete the list of main concepts of reinforcement learning, we have to specify the notion of the state S_t , as well as define the law of its evolution. This process of an agent perceiving the environment and taking actions is extended over a certain period of time. During this period, the state of the environment changes. These changes might be determined by the previous history of the environment, and in addition, can be partially driven by some random factors, as well as an agent's own actions. An immediate problem to be addressed is therefore how we model evolution of the environment. We start by describing autonomous evolution of the environment without any impact from the agent, and then generalize below to the case when such impact, i.e. a feedback loop, is present.

In a most general form, the joint probability $p_{0:T} = p(s_0, \dots, s_{T-1})$ of a particular path ($S_0 = s_0, \dots, S_{T-1} = s_{T-1}$) can be written as follows:

$$p(s_0, s_1, \dots, s_{T-1}) = \prod_{i=1}^{T-1} p(s_i | s_{0:i-1}). \quad (9.3)$$

Note that this expression does not make any assumptions about the true data-generating process—only the composition law of joint probabilities is used here. Unfortunately this expression is too general and useless in practice. This is because, for most problems of practical interest, the number of time steps encountered is in the tens or hundreds. Modeling path probabilities for sequences of states only relying on this general expression would lead to an exponential growth of the number of model parameters. We *have* to make some further assumptions in order to have a practically useful sequence model for the evolution of the environment.

The simplest, and in many practical cases a reasonable “first order approximation” approach is to additionally assume Markovian dynamics where one assumes that conditional probabilities $p(x_i | x_{0:i-1})$ depend only on K most recent values, rather than on the whole history:

$$p(s_t | s_{0:t-1}) = p(s_t | s_{t-K:t-1}). \quad (9.4)$$

The most common case is $K = 1$ where probabilities of states at time t only depend on the previously observed values of the state. Following the common tradition in the literature, unless specifically mentioned, in what follows we refer to $K = 1$ Markov processes as simply “Markov processes.” This is also the basic setting for more general Markov processes with $K > 1$: such cases can be still viewed as Markov processes with $K = 1$ but with an extended definition of a time- t state $S_t \rightarrow \hat{S}_t = (S_t, S_{t-1}, \dots, S_{t-K})$.

If we assume Markov dynamics for the evolution of the environment, this results in tractable formulations for sequence modeling. But in many practical cases, dynamics of a system cannot be as simple as a Markov process with a sufficiently low value of K , say $1 < K < 10$. For example, financial markets often have longer memories than 10 time steps. For such systems, approximating essentially non-Markov dynamics by Markov dynamics with $K \sim 10$ may not be satisfactory.

A better and still tractable way to model a non-Markov environment is to use hidden variables z_t . The dynamics is assumed to be jointly Markov in the pair (s_t, z_t) so that path probabilities factorize into the product of single-step probabilities:

$$p(s_{0:T-1}, z_{0:T-1}) = p(s_0, s_0) \prod_{t=1}^{T-1} p(z_t | z_{t-1}) p(s_t | z_t). \quad (9.5)$$

Such processes with joint Markov dynamics in a pair (s_t, z_t) of an observable and unobservable components of a state are called hidden Markov models (HMM). Note that the dynamics of the marginal x_t alone in HMMs would be in general non-Markovian.

Remarkably, introducing hidden variables z_t , we may construct models that both produce rich dynamics of observables, and have a substantially lower number of parameters than we would need with order K Markov processes. This means that such models might need considerably less data for training, and may behave better out-of-sample than Markov models. At the same time, models that are Markov in the pair (s_t, z_t) can be implemented in computationally efficient ways. Multiple examples in applications to speech and text recognition, robotics, finance demonstrated that HMMs are able to produce highly complex and sufficiently realistic sequences and time series.

The key question in any type of HMM model is how to model a hidden state z_t . Should it have a discrete or a continuous distribution? How many states are there? How should we specify the dynamics of hidden states, etc?

While these are all practically important questions, here we want to focus on the conceptual aspect of modeling. First, introduction of hidden variables z_t has a strong intuitive appeal. Traditionally, many important factors, e.g. political risk, are left outside of financial models, therefore they are “unknown” to the model. Treating such unknown risks as a de-correlated noise at each time step may be insufficient because such hidden risk factors usually include strong autocorrelations. This provides a second motivation to incorporate hidden processes in modeling of financial markets as not only a conventional tool to account for complex time dependencies of observable quantities x_t , but also on their own, as a way to account for risk factors that we do not directly include in an observable state of the model. HMMs with their Markov dynamics in the pair (x_t, z_t) provide a rather flexible set of non-Markov dynamics in observables x_t . Even richer types of non-Markov dynamics can be obtained with recurrent extensions (such as RNN and LSTM neural networks) where hidden state probabilities depend on a long history of previous hidden states rather than just on a single last hidden state.

While this implies that hidden variables might be very useful for financial machine learning, modeling decision-making of an agent in such a partially observable environment is more difficult than when an environment is fully observable. Therefore, in the rest of this chapter we will deal with models of decision-making within fully observable systems whose dynamics are assumed to be Markovian. The agent’s actions are added to the framework of Markov modeling, producing Markov Decision Process (MDP) based model. We will consider this topic next.

Example 9.2 Portfolio Trading and Reinforcement Learning

The problem of multi-period portfolio management can be described as a problem of stochastic optimal control. Consider a portfolio of stocks and a single Treasury bond, where stocks are considered risky investments with random returns, while the Treasury bond is a risk-free investment with a fixed return determined by a risk-free discount rate. Let $p_n(t)$ with $n = 1, \dots, N$ be investments at time t in N different stocks, and b_t is the investment in the

(continued)

Example 9.2 (continued)

bond. Let \mathbf{X}_t be a vector of all other relevant portfolio-specific and market-wide dynamic variables that may impact an investment decision at time t . The vector X_t may, e.g., include market prices of all stocks in the portfolio, plus market indices such as SPY500 and various sector indices, macroeconomic factors such as the inflation rate, etc.

The total state vector for such system is then $s_t = (\mathbf{p}_t, b_t, \mathbf{X}_t)$. The action \mathbf{a}_t would be an $(N + 1)$ -dimensional vector of capital allocations to all stocks and the bond at time t .

If all components of vector X_t are observable, we can model its dynamics as Markov. Otherwise, if some components of vector X_t are non-observable, we can use an HMM formulation for the dynamics. The dynamics of some components of X_t can be partially affected by actions of the trader agent, for example, market prices of stock can be moved by large trades via market impact mechanisms. A model of the interaction of the trader agent and its environment (the “market”) can therefore include feedback loop effects.

The objective of multi-step portfolio optimization is to maximize the expected cumulative reward. We can, e.g., consider one-step random rewards given by the one-step portfolio return $r_{\Pi}(s_t, a_t)$ at time step t , penalized by the one-step variance of the return: $R(s_t, a_t) = r_{\Pi}(s_t, a_t) - \lambda \text{Var}[r_{\Pi}(s_t, a_t)]$, where λ is a risk-aversion rate. Taking the expectation of this random reward, we recover the classical Markowitz quadratic reward (utility) function for portfolio optimization. Therefore, reinforcement learning with random rewards $R(s_t, a_t)$ extends Markowitz single-period portfolio optimization to a sample-based, multi-period setting.

3 Markov Decision Processes

Markov Decision Process models *extend Markov models* by adding new degrees of freedom describing *controls*. In reinforcement learning, control variables can describe agents’ actions. Controls are decided upon by the agent, and via the presence of a feedback loop, can modify the future evolution of the environment. When we embed the idea of controls with a feedback loop into the framework of Markov processes, we obtain Markov Decision Process (MDP) models.

The MDP framework provides a stylized description of goal-directed learning from interaction. It describes the agent–environment interaction as message-passing of three signals: a signal of actions by the agent, a signal of the state of an environment, and a signal defining the agent’s reward, i.e. the goal.

In mathematical terms, a Markov Decision Process is defined by a set of discrete time steps t_0, \dots, t_n and a tuple $\{\mathcal{S}, \mathcal{A}(s), p(s'|s, a), \mathcal{R}, \gamma\}$ with the following elements. First, we have a set of states \mathcal{S} , so that each observed state $S_t \in \mathcal{S}$. The

space \mathcal{S} can be either discrete or continuous. If \mathcal{S} is finite, we have a finite MDP, otherwise we have a MDP with a continuous state space.

Second, a set of actions $\mathcal{A}(s)$ defines possible actions $A_t \in \mathcal{A}(s)$ that can be taken in a state $S_t = s$. Again, the set $\mathcal{A}(s)$ can be either discrete or continuous. In the former case, we have a MDP model with a discrete action space, and in the latter case we obtain a continuous-action MDP model.

Next, a MDP is specified by transition probabilities $p(s'|s, a) = p(S_t = s' | S_{t-1} = s, a_{t-1} = a)$ of a next state S_t given a previous state S_{t-1} and an action a_{t-1} taken in this state. Slightly more generally, we may specify a joint probability of a next state s' and reward $r \in \mathcal{R}$ where \mathcal{R} is a set of all possible rewards:

$$p(s', r | s, a) = \Pr [S_t = s', R_t = r | S_{t-1} = s, a_{t-1} = a], \quad (9.6)$$

$$\sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a) = 1, \quad \forall s \in \mathcal{S}, a \in \mathcal{A}.$$

This joint probability specifies the state transition probabilities

$$p(s' | s, a) = \Pr [S_t = s' | S_{t-1} = s, a_{t-1} = a] = \sum_{r \in \mathcal{R}} p(s', r | s, a), \quad (9.7)$$

as well as the *expected reward function* $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ that gives the expected value of a random reward R_t received in the state $S_{t-1} = s$ upon taking action $a_{t-1} = a$:

$$r(s, a, s') = \mathbb{E} [R_t | S_{t-1} = s, A_{t-1} = a, S_t = s']$$

$$= \sum_{r \in \mathcal{R}} r \frac{p[S_t = s', R_t = r | S_{t-1} = s, a_{t-1} = a]}{p[S_t = s' | S_{t-1} = s, a_{t-1} = a]}. \quad (9.8)$$

Finally, a MDP needs to specify a *discount factor* γ which is a number between 0 and 1. We need the discount factor γ to compute the *total cumulative reward* given by the sum of all single-step rewards, where each next term gets an extra power of γ in the sum:

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \dots \quad (9.9)$$

This means that as long as $\gamma < 1$, receiving a larger reward now and a smaller reward later is preferred to receiving a smaller reward now and a larger reward later. The discount factor γ just controls by how much the first scenario is preferable to the second one. This means that the discount factor for a MDP plays a similar role to a discount factor in finance, as it reflects the time value of rewards. Therefore, in financial application the discount factor γ can be identified with a continuously-

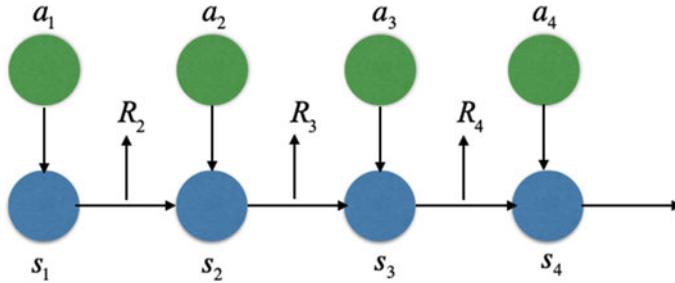


Fig. 9.1 The causality diagram for a Markov Decision Process

compounded interest rate. Note that for infinite-horizon MDPs, a value $\gamma < 1$ is required in order to have a finite total reward.²

A diagram describing a Markov Decision Process is shown in Fig. 9.1. The blue circles show the evolving state of the system S_t at discrete-time steps. These states are connected by arrows representing *causality* relations. We have only one arrow that enters each blue circle from a previous blue circle, emphasizing a Markov property of the dynamics: each next state depends only on the previous state, but *not* on the whole history of previous states. The green circles denote actions A_t taken by the agent. The upwards pointing arrow denote rewards R_t received by the agent upon taking actions A_t .

The goal in a Markov Decision Process problem or in reinforcement learning is to maximize the expected total cumulative reward (9.9). This is achieved by a proper choice of a *decision policy* that specifies how the agent should act in each possible state.

3.1 Decision Policies

Let us now consider how we can actually achieve the goals of maximizing the expected total rewards for Markov Decision Processes. Recall that the goal of reinforcement learning is to maximize the expected total reward from all actions performed in the future. Because this problem has to be solved *now*, but actions will be performed in the *future*, to solve this problem, we have to define a “policy”.

A policy $\pi(a|s)$ is a function that takes a current state $S_t = s$, and translates it into an action $A_t = a$. In other words, this function *maps* a state space onto an action space in the Markov Decision Process. If a system is currently in a state described by a vector S_t , then the next action A_t is given by a *policy function* $\pi(S_t)$. If the

²An alternative formulation for infinite-horizon MDPs is to consider maximization of an average reward rather than total reward. Such approach allows one to proceed without introducing a discount factor. We will not pursue reinforcement learning with average rewards in this book.

policy function is a conventional function of its argument S_t , then the output A_t will be a single number. For example, if the policy function is $\pi(S_t) = 0.5S_t$, then for each possible value of S_t we will have one action to take. Such specification of a policy is called a *deterministic policy*.

Another way to analyze policies in MDPs is to consider *stochastic policies*. In this case, a policy $\pi(a|s)$ describes a probability distribution rather than a function. For example, suppose that there are two actions a_0 and a_1 , then the stochastic policy might be given by the logistic function $\pi_0 := \pi(a = a_0|s) = \sigma(\theta^T s) = \frac{1}{1 + \exp(-\theta^T s)}$ and $\pi_1 := \pi(a = a_1|s) = 1 - \pi_0$.

Now we take a look at differences between these two specifications in slightly more detail.

First, consider deterministic policies. In this case, the action A_t to take is given by the value of a deterministic policy function $\pi(S_t)$ applied to a current state S_t . If the RL agent finds itself in the same state S_t of the system more than once, each time it will act exactly the same way. How it will act depends only on the current state S_t , but *not* on any previous history. This assumption is made to ensure consistency with the Markov property of the system dynamics, where probabilities to observe specific future states depend only on the current state, but not on any previous states. It can actually be proven that an optimal deterministic policy π always exists for a Markov Decision Process, so our task is simply to identify it among all possible deterministic policies.

As long as that is the case, it might look like deterministic policies is *all* we ever need to solve Markov Decision Processes. But it turns out that the second class of policies, namely *stochastic policies*, are also often useful for reinforcement learning.

For stochastic policies, a policy $\pi(a|s)$ becomes a *probability distribution* over possible actions $A_t = a$. This distribution may depend on the current value of $S_t = s$ as a parameter of such distribution. So, if we use a stochastic policy instead of a deterministic policy, when an agent visits the same state again, it might take a different action from an action it took the last time in this state. Note that the class of stochastic policies is wider than the class of deterministic policies, as the latter can always be thought as limits of stochastic policies where a distribution collapses to a Dirac delta-function. For example, if a stochastic policy is given by a Gaussian distribution with variance σ^2 , then a deterministic Dirac-like policy can be obtained in this setting by taking the limit $\sigma^2 \rightarrow 0$.

We might consider stochastic policies as a more general specification, but why would we want to consider such stochastic policies? It turns out that in certain cases, introducing stochastic policies might be an unnecessary complication. In particular, if we *know* transition probabilities in the MDP, then we can just consider only deterministic policies to find an optimal deterministic policy. For this, we just need to solve the Bellman equation that we introduce in the next section.

But if we do *not* know the transition probabilities, we must either estimate them from data and use them to solve the Bellman equation, or we have to rely on *samples*, following the reinforcement learning approach. In this case, randomization

of possible actions following some stochastic policy may provide some margin for *exploration*—for a better estimation of the model.

A second case when a stochastic policy may be useful is when instead of a fully observed (Markov) process, we use a partially observed dynamic model of an environment which can be implemented as, e.g. a HMM process. Introducing additional control variables in this model would produce a Partially Observable Markov Decision Process, or POMDP for short. Stochastic policies may be optimal for POMDP processes, which is different from a fully observable MDP case where the best policy can always be found within the class of deterministic policies.

While many problems of practical interest in finance would arguably lend themselves to a POMDP formulation, we leave such formulations outside the scope of this book where we rather focus on reinforcement learning for fully observed MDP settings.

3.2 Value Functions and Bellman Equations

As mentioned above, a common auxiliary task of learning by acting is to evaluate a given state of the environment. Because it is defined by a sequence of actions taken by the agent, we want to relate it to local rewards obtained by the agent. One way to define the value function $V^\pi(s)$ for policy π is to specify it as an expected total reward that can be obtained starting from state $S_t = s$ and following policy π :

$$V_t^\pi(s) = \mathbb{E}_t^\pi \left[\sum_{i=0}^{T-t-1} \gamma^i R(S_{t+i}, a_{t+i}, S_{t+i+1}) \middle| S_t = s \right], \quad (9.10)$$

where $R(S_{t+i}, a_{t+i}, S_{t+i+1})$ is a random future reward at time $t + i$, T is the planning time horizon (an infinite-horizon case corresponds to $T = \infty$), and $\mathbb{E}_t^\pi[\cdot | S_t = s]$ means time- t averaging (conditional expectation) over all future states of the world given that future actions are selected according to policy π .

The value function (9.10) can be used for two main settings for reinforcement learning. For *episodic tasks*, the problem of the agent learning naturally breaks into *episodes* which can be either of fixed or variable length $T < \infty$. For such tasks, using a discount factor γ for future rewards is not strictly necessary, and we could set there $\gamma = 1$ if so desired. On the other hand, for *continuing tasks*, the agent–environment interaction does not naturally break into episodes. Such tasks corresponds to the case $T = \infty$ in (9.10), which makes it necessary to keep $\gamma < 1$ to ensure convergence of an infinite series of rewards.

The *state-value function* $V_t^\pi(s)$ is therefore given by a conditional expectation of the total cumulative reward, also known in reinforcement learning as the random *return* G_t :

$$G_t = \sum_{i=0}^{T-t-1} \gamma^i R(S_{t+i}, a_{t+i}, S_{t+i+1}). \quad (9.11)$$

For each possible state $S_t = s$, the state-value function $V^\pi(s)$ gives us the “value” of this state for the task of maximizing the total reward using policy π . The state-value function $V_t^\pi(s)$ is thus a *function* of s and a *functional* (i.e., a function of a function) of a decision policy π . For time-homogeneous problems, the time index can be omitted, $V_t^\pi(s) \rightarrow V^\pi(s)$.

Similarly, we can specify a value of simultaneously being in state s and take action a as a first action, and following policy π for all the following actions. This defines the *action-value function* $Q^\pi(s, a)$:

$$Q_t^\pi(s, a) = \mathbb{E}_t^\pi \left[\sum_{i=0}^{T-t-1} \gamma^i R(S_{t+i}, a_{t+i}, S_{t+i+1}) \middle| S_t = s, a_t = a \right]. \quad (9.12)$$

Note that in comparison to the state-value function $V_t^\pi(s)$, the state-value function $Q^\pi(s, a)$ depends on an additional argument a which is an action taken at time step t . While we may consider arbitrary inputs $a \in \mathcal{A}$ to the action-value function $Q^\pi(s, a)$, it is of particular interest to consider a special case when the first action a is also drawn from policy $\pi(a|s)$. In this case, we obtain a relation between the state-value function and the action-value function:

$$V_t^\pi(s) = \mathbb{E}^\pi [Q_t^\pi(s, a)] = \sum_{a \in \mathcal{A}} \pi(a|s) Q_t^\pi(s, a). \quad (9.13)$$

(Note that while we assume a finite MDP formulation in this section, the same formulas applied to continuous-state MDPs model provided by replace sums by integrals).

Both the state-value function $V_t^\pi(s)$ and action-value function $Q_t^\pi(s, a)$ are given by conditional expectations of the return G_t from states. Therefore, they could be estimated directly from data if we have observations of total returns G_t obtained in different states. The simplest case arises for a finite MDP model. Let us assume that we have K discrete states, and for each state, we have data referring observed sequences of states, actions, and rewards obtained while starting in each one of the K states. Then values functions in different states can be estimated empirically using these observed (or simulated) sequences. Such methods are referred to in reinforcement learning literature as Monte Carlo methods.

Another useful way to analyze value functions is to rely on their particular structure defined as an expectation of cumulative future rewards. As we will see next, this can be used to derive certain equations for the state-value function (9.10) and action-value function (9.12).

Consider first the state-value function (9.10). We can separate the first term from the sum, and write the latter as the first term plus a similar sum but starting with the second term:

$$V_t^\pi(s) = \mathbb{E}_t^\pi [R(s, a_t, S_{t+1})] + \gamma \mathbb{E}_t^\pi \left[\sum_{i=1}^{T-t-1} \gamma^i R(S_{t+i}, a_{t+i}, S_{t+i+1}) \mid S_t = s \right].$$

Note that the first term in the right-hand side of this equation is just the expected reward from the current step. The second term is the same as the expectation of the value function at the next time step $t + 1$ in some other state s' , multiplied by the discount factor γ . This gives

$$V_t^\pi(s) = \mathbb{E}_t^\pi [R_t(s, a, s')] + \gamma \mathbb{E}_t^\pi [V_{t+1}^\pi(s')]. \quad (9.14)$$

Equation (9.14) then gives an expression for a value of the state as a sum of immediate expected reward and a discounted expectation of the expected next state value. Note further that the conditional time- t expectations $\mathbb{E}_t^\pi [\cdot]$ in this expression involve conditioning on the current state $S_t = s$.

Equation (9.14) thus presents a simple recursive scheme that enables computation of the value function at time t in terms of its future values at time $t + 1$ by going backward in time, starting with $t = T - 1$. This relation is known as the “Bellman equation for the value function”. Later on we will introduce a few more equations that are also called Bellman equations. It was proposed in 1950s by Richard Bellman in the context of his pioneering work on dynamic programming. Note that this is a linear equation, because the second expectation term is a linear functional of $V_{t+1}^\pi(s')$. In particular, for a finite MDP with N distinct states, the Bellman equation produces a set of N linear equations defining the value function at time t for each state in terms of expected immediate rewards, and transition probabilities to states at time $t + 1$ and next-period value functions $V_{t+1}^\pi(s')$ that enter the expectation in its right-hand side. If transition probabilities are known, we can easily solve this linear system using methods of linear algebra.

Finally, note that we could repeat all steps leading to the Bellman equation (9.14), but starting with the action-value function rather than the state-value function. This produces the Bellman equation for the action-value function:

$$Q_t^\pi(s, a) = \mathbb{E}_t^\pi [R_t(s, a, s')] + \gamma \mathbb{E}_t^\pi [V_{t+1}^\pi(s')]. \quad (9.15)$$

Similarly to (9.14), this is a linear equation that can be solved backward in time using linear algebra for a finite MDP model.

➤ Learning with a Finite vs Infinite Horizon

While an MDP problem is formulated in the same way for both cases of a finite $T < \infty$ or infinite $T \rightarrow \infty$ time horizon, computationally they produce different algorithms. For infinite-horizon MDP problems with rewards that do not explicitly depend on time, a state- and action-value function should also not depend explicitly on time: $V_t(s_t) \rightarrow V(s_t)$ and similarly for $G_t(s_t, a_t) \rightarrow G(s_t, a_t)$, which expresses time-invariance of such problem. For time-invariant problems, the Bellman equation (9.15) becomes a fixed-point equation for the *same* function $Q(s, t)$ rather than a recursive relation between different functions $Q_t(s_t, a_t)$ and $Q_{t+1}(s_{t+1}, a_{t+1})$.

While many of existing textbooks and other resources are often focused on presenting and pursuing algorithms for MDP and RL for a time-independent setting, including analyses of convergence, a proper question to ask is which one of the two types of MDP problem is more relevant for financial applications? We tend to suggest that finite-horizon MDP problems are more common in finance, as most of goals in quantitative finance focus on performance over some pre-specified time horizon such as one day, one month, one quarter, etc. For example, an annual performance time horizon of $T = 1Y$ for a mutual fund or an ETF sets up a natural time horizon for planning in the MDP formulation. On the other hand, a given fixed time horizon may consist of a large number of smaller time steps. For example, a daily fund performance can result from multiple trades executed on a minute scale during the day. Alternatively, multi-period optimization of a long-term retirement portfolio with a time horizon of 30 years can be viewed at the scale of monthly or quarterly steps. For such cases, it may be reasonable to approximate an initially time-dependent problem with a fixed but large number of time steps by a time-independent infinite-horizon problem. The latter is obtained as an approximation of the original problem when the number of time steps goes to infinity. Therefore, an infinite-horizon MDP formulation can serve as a useful approximation for problem involving long sequences.

3.3 Optimal Policy and Bellman Optimality

Next, we introduce an *optimal* value function V_t^* . The optimal value function for a state is simply the highest value function for this state among all possible policies. So, the optimal value function is attained for some *optimal policy* that we will call π_* . The important point here is that an optimal policy π_* is optimal for *all* states of the system. This means that V_t^* should be larger or equal than V_t^π with *any* other

policy π , and for *any* state S , i.e. $\pi_\star > \pi$ if $V_t^\star := V_t^{\pi_\star}(s) \geq V_t^\pi(s), \forall s \in S$. We may therefore express V_\star as follows:

$$V_t^\star(s) := V_t^{\pi_\star}(s) = \max_{\pi} V_t^\pi(s), \quad \forall s \in S. \quad (9.16)$$

Equivalently, the optimal policy π_\star can be determined in terms of the action-value function:

$$Q_t^\star(s, a) := Q_t^{\pi_\star}(s, a) = \max_{\pi} Q_t^\pi(s, a), \quad \forall s \in S. \quad (9.17)$$

This function gives the expected reward from taking action a in state s , and following an optimal policy thereafter. The optimal action-value function can therefore be represented by the following Bellman equation that can be obtained from Eq. (9.15):

$$Q_t^\star(s, a) = \mathbb{E}_t^\star [R_t(s, a, s')] + \gamma \mathbb{E}_t^\star [V_{t+1}^\star(s')]. \quad (9.18)$$

Note that this equation might be inconvenient to work with in practice, as it involves two optimal functions $Q_t^\star(s, a)$ and $V_t^\star(s)$, rather than values of the same function at different time steps, as in the Bellman equation (9.14). However, we can obtain a Bellman equation for the optimal action-value function $Q_t^\star(s, a)$ in terms of this function only, if we use the following relation between two optimal value functions:

$$V_t^\star(s) = \max_a Q_t^\star(s, a). \quad (9.19)$$

We can now substitute Eq. (9.19) evaluated at time $t + 1$ into Eq. (9.18). This produces the following equation:

$$Q_t^\star(s, a) = \mathbb{E}_t^\star \left[R_t(s, a, s') + \gamma \max_{a'} Q_{t+1}^\star(s', a') \right]. \quad (9.20)$$

Now, unlike Eq. (9.18), this equation relates the optimal action-value function to its values at a later time moment. This equation is called the Bellman optimality equation for the action-value function, and it plays a key role in both dynamic programming and reinforcement learning. This expresses the famous Bellman's principle of optimality, which states that optimal cumulative rewards should be obtained by taking an optimal action now, and following an optimal policy later.

Note that unlike Eq. (9.18), the Bellman optimality equation (9.20) is a non-linear equation, due to the max operation inside of the expectation. Respectively, the Bellman optimality equation is harder to solve than the linear Bellman equation (9.14) that holds for an arbitrary fixed policy π . The Bellman optimality equation is usually solved numerically. We will discuss some classical methods of solving it in the next sections.

The Bellman equation for the optimal state-value function $V_t^*(s)$ can be obtained using Eqs. (9.18) and (9.19):

$$V_t^*(s) = \max_a \mathbb{E}_t^* [R_t(s, a, s') + \gamma V_{t+1}^*(s')]. \quad (9.21)$$

Like Eq. (9.20), the Bellman optimality equation (9.21) for the state-value function is a non-linear equation due to the presence of the max operator.

If the optimal state-value or action-value function is already known, then finding the optimal action is simple, and essentially amounts to “greedy” one-step maximization. The term “greedy” is used in computer science to describe algorithms that are based only on intermediate (single-step) considerations but not on considerations of longer-term implications. If we already know the optimal state-value function $V_t^*(s)$, this implies that all actions at all time steps implied in this value function are already optimal. This still does not determine on its own what action should be chosen at a current time step. However, because we know that actions at the subsequent steps are already optimal, to find the best next action in this setting we need only perform a greedy one-step search that just takes into account the immediate successor states for the current state. In other words, when the optimal state-value function $V_t^*(s)$ is used, a one-step-ahead search for optimal action produces long-term optimal actions.

With the optimal action-value function $Q_t^*(s, a)$, the search of an optimal action at the current step is even simpler, and amounts to simply maximizing $Q_t^*(s, a)$ with respect to a . This does not require using any information about possible successor states and their values. This means that all relevant information of the dynamics is already encoded in $Q_t^*(s, a)$.

The Bellman optimality equations (9.20) and (9.21) are central objects for decision-making under the formalism of MDP models. The methods of dynamic programming focus on exact or numerical solutions of these equations. Many (though not all) methods of reinforcement learning are also based on approximate solutions to the Bellman optimality equations (9.20) and (9.21). As we will see in more details later on, the main difference of these methods from traditional dynamic programming methods is that they rely on empirically observed transitions rather than on a theoretical model of transitions.

► Existence of Bellman Equations: Infinite-Horizon vs Finite Horizon Cases

For time-homogeneous (infinite horizon) problems, a value function does not explicitly depend on time, and the Bellman equation (9.14) can be written in this case in a compact form

$$T^\pi V^\pi = V^\pi, \quad (9.22)$$

(continued)

where $T^\pi : \mathbb{R}^S \rightarrow \mathbb{R}^S$ stands for the *Bellman operator*

$$(T^\pi V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s', s, \pi(s))V(s'), \quad \forall s \in S. \quad (9.23)$$

Therefore, for time-stationary MDP problems the Bellman equation becomes a fixed-point equation. If $0 < \gamma < 1$, then T^π is a maximum-norm contraction and the fixed-point equation (9.22) has a unique solution, see, e.g., Bertsekas (2012).

Similarly, the Bellman optimality equation (9.21) can be written in a time-stationary case as

$$T^*V^* = V^*, \quad (9.24)$$

where $T^* : \mathbb{R}^S \rightarrow \mathbb{R}^S$ is for the *Bellman optimality operator*

$$(T^*V)(s) = \max_{a \in \mathcal{A}} \left\{ r(s, a) + \gamma \sum_{s' \in S} p(s', s, a)V(s') \right\}, \quad \forall s \in S. \quad (9.25)$$

Again, if $0 < \gamma < 1$, then T^* is a maximum-norm contraction and the fixed-point equation (9.22) has a unique solution, see Bertsekas (2012).

For finite-horizon MDP problem the Bellman equations (9.14) and (9.21) become recursive relations between different functions $V_t^\pi(s_t)$ and $V_{t+1}^\pi(s_{t+1})$. The existence of a solution for this case can be established by mapping the finite-horizon MDP problem onto a stochastic shortest-path (SSP) problem. SSP problems have a certain terminal (absorbing) state, and the problem of an agent is to incur a minimum expected total cost on a path to the terminal state. For details on existence of the Bellman equation for the SSP, see Vol II of Bertsekas (2012). In a finite-horizon MDP, time t can be thought of as a *stage* of the process, such that after N stages the system enters the absorbing state with probability one. The finite-horizon problem is therefore mapped onto a SSP problem with an augmented state $\tilde{s}_t = (s_t, t)$.

4 Dynamic Programming Methods

Dynamic programming (DP), pioneered by Bellman (1957), is a collection of algorithms that can be used to solve problems of finding optimal policies when an environment is Markov and fully observable, so that we can use the formalism of Markov Decision Processes. Dynamic programming approaches work for finite MDP models with typically a low number of states, and moreover assume that the

model of the environment is perfectly known. Both cases are rarely encountered in problems of practical interest where a model of the environment is typically not known beforehand, and a state space is often either discrete and high-dimensional, or continuous, or continuous with multiple dimensions. Methods of DP which find exact solutions for finite MDP become infeasible for such situations.

Still, while they are normally not very useful for practical problems, methods of DP have fundamental importance for understanding other methods that *do* work in “real-world” applications and are applied in reinforcement learning, as well as in other related approaches such as approximate dynamic programming. For example, we may use DP methods to validate RL algorithms applied to simulated data with known probability transition matrix. Here we want to analyze the most popular DP algorithms.

A common feature in all these algorithms is that they all rely on the notion of a state-value function (or action-value function, or both) as a condensed representation of quality of both policies and states. Respectively, extensions of such algorithms to (potentially high-dimensional) sample-based approached performed by RL methods are generically called the value function based RL.

Some RL methods do not rely on the notion of a value function, and operate only with policies. In this book, we will focus mostly on the value function based RL but we shall occasionally see examples of an alternative approach based on “policy iteration.”

In addition, all approaches considered in this section imply that a state-action space is discrete. While usually not explicitly specified, the dimensionality of a state space is assumed to be sufficiently low, so that the resulting computational algorithm would be feasible in practice. DP approaches are also used for systems with a continuous state by discretizing a range of values. For a multi-dimensional continuous state space, we could discretize each individual component, and then produce a discretized uni-dimensional representation by taking a direct product of individual grids, and indexing all resulting states. However, this approach is straightforward and feasible when the number of continuous dimensions is sufficiently low, e.g. do not exceed 3 or 4. For higher dimensional continuous problems a naive discretization by forming cross-products of individual grids produces an exponential explosion of a number of discretized steps, and quickly becomes infeasible in practice.

4.1 Policy Evaluation

As we mentioned above, the state-value function $V_t^\pi(s)$ gives the value of the current state $S_t = s$ provided the agent follows policy π in choosing its actions. For a time-stationary MDP, which will be considered in this section, the time-independent version of the Bellman equation (9.14) reads

$$V^\pi(s) = \mathbb{E}_t^\pi [R(s, a_t, S_{t+1}) + \gamma V^\pi(s')], \quad (9.26)$$

while the time-independent version of the Bellman optimality equation (9.21) is

$$V^*(s) = \max_a \mathbb{E}_t^* [R(s, a, s') + \gamma V^*(s')]. \quad (9.27)$$

Thus, for time-stationary problems, the problem of finding the state-value function $V^\pi(s)$ for a given policy π amounts to solving a set of $|\mathcal{S}|$ linear equations, where $|\mathcal{S}|$ stands for the dimensionality of the state space. As solving such a system directly (in one step) involves matrix inversion, such a solution can be costly when the dimensionality $|\mathcal{S}|$ of the state-space grows.

As an alternative, the Bellman equation (9.26) can be used to set up a recursive approach to solve it. While this produces a multi-step numerical algorithm to find the state-value function rather than an explicit one-step formulas obtained with the linear algebra approach, in practice it often works better than the latter.

The idea is simply to view the Bellman equation (9.26) as a stationary point at convergence as $k \rightarrow \infty$ of an iterative map indexed by steps $k = 0, 1, \dots$, which is obtained by applying the Bellman operator

$$\mathcal{T}^\pi [V] := \mathbb{E}_t^\pi [R(s, a_t, S_{t+1}) + \gamma V(s')], \quad (9.28)$$

to the previous iteration $V_k^{(\pi)}(s)$ of the value function. Here the lower index k enumerates iterations, and replaces the time index t which is omitted because we work in this section with time-homogeneous MDPs. This produces the following update rule:

$$V_k^\pi(s) = \mathbb{E}_t^\pi [R(s, a_t, S_{t+1}) + \gamma V_{k-1}^\pi(s')]. \quad (9.29)$$

The informal way to understand this relation is to think about a recursion as a sequential process. The sequential nature of the process can be mapped onto some notion of time. Therefore, if we formally replace $T - t \rightarrow k$ in the original time-dependent Bellman equation (9.14), it produces Eq. (9.29). The relation (9.29) is often referred to as the *Bellman iteration*. The Bellman iteration is proven to converge under some technical conditions on the Bellman operator (9.28). In particular, rewards need to be bounded to guarantee convergence of the map.

For any given policy π , *policy evaluation* algorithm amounts to a repeated application of the Bellman iteration (9.29) starting with some initial guess, e.g. $V_0^{(\pi)}(s) = 0$. The iteration continues until convergence at a given tolerance level is achieved, or alternatively it can run for a pre-specified number of steps. Each iteration involves only linear operations, therefore it can be performed very fast. Recall here that the model of the environment is assumed to be perfectly known in the DP approach, therefore all expectations are linear and fast to compute. The method is scalable to high-dimensional discrete state spaces.

While a policy evaluation algorithm can be quite fast in evaluating a single policy π , the ultimate goal of both dynamic programming and reinforcement learning approaches is to find the optimal policy π_* . This opens the door to a plethora of different approaches. In one class of approaches, we consider a set of candidate

policies $\{\pi\}$ and we find the policy π_* by selecting between them. These methods are called “policy iteration” algorithms, and they use the policy evaluation algorithms as their integral part. We will consider such algorithms next.

4.2 Policy Iteration

Policy iteration is a classical algorithm for finite MDP models. Again, we consider here only stationary problems, therefore we again replace the time index by an iteration index $V_{T-t}^\pi(s) \rightarrow V_k^*(s)$. We can also omit the index π here as the purpose of this algorithm is to find the optimal $\pi = \pi_*$.

To produce a policy iteration algorithm, we need two features: a way to evaluate a given policy and a way to improve a given policy. Both can be considered sub-algorithms in an overall algorithm. Note that the first component in this scheme is already available, and is given by the policy evaluation method just presented. Therefore, to have a complete algorithms, our only remaining task is to supplement this with a method to improve a given policy.

To this end, consider the Bellman equation for the action-value function

$$Q^\pi(s, a) = \mathbb{E}_t [R(s, a, s') + \gamma V^\pi(s')] = \sum_{s', r} p(s', r | s, a) [R(s, a, s') + \gamma V^\pi(s')]. \quad (9.30)$$

The current action a is a control variable here. If we follow policy π in choosing a , then the action taken is $a = \pi(s)$, and the action value is $Q^\pi(s, \pi(s)) = V^\pi(s)$. This means that by taking *different* actions $a \sim \pi'$ (with another policy π') rather than prescribed by policy π , we can produce higher values $Q^\pi(s, \pi'(s))$. It can be shown that this implies that the new policy also improves the state-value function, i.e. $V^{\pi'}(s) \geq V^\pi(s)$ for all states $s \in \mathcal{S}$ (the latter statement is called the policy improvement theorem, see, e.g., Sutton and Barto (2018) or Szepesvari (2010) for details.)

Now imagine we want to choose action a in Eq. (9.30) so that to maximize $Q^\pi(s, a)$. Maximizing $a \rightarrow a_*$ means that we find a value a_* such that $Q^\pi(s, a_*) \geq Q^\pi(s, a)$ for any $a \neq a_*$. This can be equivalently thought of as a greedy policy π' that is given by π except for the state s , where it should be such that $a_* = \pi'(s)$. If some greedy policy $\pi' \neq \pi$ can be found by maximizing $Q^\pi(s, a)$, or equivalently the right-hand side of Eq. (9.30), then it will satisfy the policy improvement theorem, and then can be used to find the optimal policy via policy iteration. Therefore, an inexpensive search for a better greedy policy π' by a local optimization over possible next actions $a \in \mathcal{A}$ is guaranteed to produce a sequence of policies that are either better than the previous ones, or at worst keep them unchanged.

This observation underlies the policy iteration algorithm which proceeds as follows. We start with some initial policy $\pi^{(0)}$. Often, a purely random initialization is used. After that, we repeat the following set of two calculations, for a fixed number of steps or until convergence:

- *Policy evaluation*: For a given policy $\pi^{(k-1)}$, compute the value function $V^{(k-1)}$ by solving the Bellman equation (9.26)
- *Policy improvement*: Calculate new policy

$$\pi^{(k)} = \arg \max_{a \in \mathcal{A}} \sum_{s'} p(s'|s, a) \left[R(s, a, s') + \gamma V^{(k-1)}(s') \right]. \quad (9.31)$$

In words, at each iteration step we first compute the value function using the previous policy, and then update the policy using the current value function. The algorithm is guaranteed to converge for a finite state MDP with bounded rewards.

Note that if the dimensionality of the state space is large, multiple runs of policy evaluation can be quite costly, because it would involve high-dimensional systems of linear equations. But many practical problems of optimal control involve large discrete state-action spaces, or continuous state-action space. In these settings, methods of DP introduced by Bellman (1957), and algorithms like policy iteration (or value iteration, to be presented next), do *not* work anymore. Reinforcement learning methods were developed in particular as a practical answer to such challenges.

4.3 Value Iteration

Value iteration is another classical algorithm for finite and time-stationary MDP models. Unlike the policy iteration method, it bypasses the policy improvement stage, and uses a recursive procedure to directly find the optimal state-value function $V^*(s)$.

The value iteration method works by applying the Bellman optimality equation as an update rule in an iterative scheme. In more detail, we start with initialization of the value function at some initial values $V(s) = V^{(0)}(s)$ for all states, with some choice of function $V^{(0)}(s)$, e.g. $V^{(0)}(s) = 0$. Then we continue iterating the evaluation of the value function using the Bellman optimality equation as the definition of the update rule. That is, for each iteration $k = 1, 2, \dots$, we use the result of the previous iteration to compute the right-hand side of the equation:

$$V^{(k)}(s) = \max_a \mathbb{E}_t^* \left[R(s, a, s') + \gamma V^{(k-1)}(s') \right] = \max_a \sum_{s', r} p(s', r | s, a) \left[r + \gamma V^{(k-1)}(s') \right]. \quad (9.32)$$

This can be thought of as combining the two steps of policy improvement and policy evaluation into one update step. Note that the new value iteration update rule (9.32) is similar to the policy evaluation update (9.29) except that it also involves taking a maximum over all possible actions $a \in \mathcal{A}$.

Now, there are a few ways to update the value function in such a value iteration. One approach is to complete re-computing the value function for all states $s \in \mathcal{S}$,

and then simultaneously update the value function over all states, $V^{(k-1)}(s) \rightarrow V^{(k)}(s)$. This is referred to as *synchronous* updating.

The other approach is to update the value function $V^{(k-1)}(s)$ on the fly, as it is re-computed in the current iteration k . This is called *asynchronous* updating. Asynchronous updates are often used for problems with large state-action spaces. When only a relatively small number of states matter for an optimal solution, updating all states after a complete sweep, as is done with synchronous updates, might be inefficient for high-dimensional state-action spaces. For either way of updating, it can be proven that the algorithm converges to the optimal value function $V^*(s)$. After $V^*(s)$ is found, the optimal policy π_* can be found using the same formula as before.

As one can see, the basic algorithm is very simple, and works well, as long your state-action space is discrete and has a small number of states. However, similarly to policy iteration, the value iteration algorithm quickly becomes unfeasible in high-dimensional discrete or continuous state spaces, due to exponentially large memory requirements. This is known as the *curse of dimensionality* in the DP literature.³ Given that the time needed for DP solutions to be found is polynomial in the number of states and actions, this may also produce prohibitively long computing times.⁴

For low-dimensional continuous state-action spaces, the standard approach enabling applications of DP is to discretize variables. This method can be applied only if the state dimensionality is very low, typically not exceeding three or four. For higher dimensions, a simple enumeration of all possible states leads to an exponential growth of the number of discretized states, making the classical DP approaches unfeasible for such problems due to memory and speed limitations. On the other hand, as will be discussed in details below, the RL approach relies on samples, which are always discrete-valued even for continuous distributions. When a sampling-based approach of RL is joined with some reasonable choices for a low-dimensional bases in such continuous spaces (i.e., using some methods of function approximation), RL is capable of working with continuously valued multi-dimensional states and action.

Recall that DP methods aim for a numerically exact calculation of optimal value function at all points of a discrete state space. However, what is often needed in high-dimensional problems is an *approximate* way of computing the value functions using simultaneously a lower, and often much lower, number of parameters than the original dimensionality of the state space. Such methods are called *approximate dynamic programming*, and can be applied in situations when the model of the world is known (or independently estimated beforehand from data), but the dimensionality of a (discretized) state space is too large to apply the standard value or policy

³While high dimensionality is a curse for DP approaches as it makes them infeasible for high-dimensional problems, with some other approaches this may rather bring simplifications, in which case the “curse of dimensionality” is replaced by the “blessing of dimensionality.”

⁴Computing times that are polynomial in the number of states and actions are obtained for worst-case scenarios in DP. In practical applications of DP, convergence is sometimes faster than constraints given by worst-case scenarios.

iteration methods. On the other hand, the reinforcement learning approach works directly with samples from data. When it is combined with a proper method of function approximation to handle a high-dimensional state space, it provides a sample-based RL approach to optimal control—which is the topic we will pursue next.

Example 9.3 Financial Cliff Walking

Consider an over-simplified model of household finance. Let S_t be the amount of money the household has in a bank account at time t . We assume for simplicity that S_t can only take values in a discrete set $\{S^{(i)}\}_{i=0}^{N-1}$. The account has to be maintained for T time steps, after which it should be closed, so T is the planning horizon. The zero level $S^{(0)} = 0$ is a bankruptcy level—it has to be avoided, as reaching it means inability to pay on household's liabilities. At each step, the agent can deposit to the account $S^{(i)} \rightarrow S^{(i+1)}$ (action a_+), withdraw from the account $S^{(i)} \rightarrow S^{(i-1)}$ (action a_-), or keep the same amount $S^{(i)} \rightarrow S^{(i)}$ (action a_0). The initial amount in the account is zero. For any step before the final step T , if the agent moves to the zero level $S_0 = 0$, it receives a negative reward of -100 , and the episode terminates. Otherwise, the agent continues for all T steps. Any action not leading to the zero level gets a negative reward of -1 . At the terminal time, if the final state is $S_T > 0$, the reward is -1 , but if the account goes back to zero exactly at time T , i.e. $S_T = 0$, the last action gets a positive reward of $+10$. The learning task is to maximize the total reward over T time steps (Fig. 9.2). The RL agent has to learn the optimal depository policy online, by trying different actions during a training episode. Note that while this is a time-dependent problem, we can map it onto a stationary problem with an episodic task and a target state, such as the original cliff walking problem in Sutton and Barto (2018).

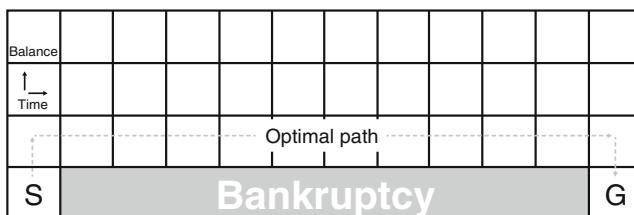


Fig. 9.2 The financial cliff walking problem is closely based on the famous cliff walking problem by Sutton and Barto (2018). Given a bank account with an initial zero balance (“start”), the objective is to deposit and deplete the account by a unit of currency so that the account ends with a zero balance at the final time step (“goal”). Premature depletion is labeled as a bankruptcy and terminates the game. Reaching the final time with a surplus amount in the account results in a penalty. At each time step, the agent may choose from a number of actions if the account is not zero: deposit (“U”), withdraw (“D”), or do nothing (“Z”). Transactions costs are imposed so that the optimal policy is to hold the balance at unity

5 Reinforcement Learning Methods

Reinforcement learning methods aim at the same goal of solving MDP models as do the DP methods. The main differences are in how the problems of data processing and computational design are approached. This section provides a brief overview of some of the most popular reinforcement learning methods for solving MDP problems.

Three main classes of approaches to reinforcement learning are Monte Carlo methods, policy search methods, and value-based RL. The first two methods do not rely on Bellman equations, and thus do not have direct links to the Bellman equation introduced in this chapter. While we present a brief overview of Monte Carlo and policy search methods, most of the material presented in the later chapters of this book uses value-based RL. In what follows in later sections, we will often refer to value-based RL as simply “RL”.

As we just mentioned, these RL approaches use the Bellman optimality equations (9.20) and (9.21); however, they proceed differently. With DP approaches, one attempts to solve these equations exactly. This is only feasible if a perfect model of the world is known and the dimensionality of the state-action space is sufficiently low. As we remarked earlier, both assumptions do not hold in most problems of practical interest.

Reinforcement learning approaches do not assume that a perfect model of the world is known. Instead, they simply rely on actual data that are viewed as samples from a true data-generating distribution. The problem of estimating this distribution can be bypassed altogether with reinforcement learning. In particular, *model-free* reinforcement learning operates directly with samples of data, and relies only on samples when optimizing its policy. This is not to say, of course, that models of the future are useless for reinforcement learning. *Model-based* reinforcement learning approaches build an internal model of the world as a part of their ultimate goal of policy optimization. We will discuss model-based reinforcement learning later in this book, but in this section, we will generally restrict consideration to model-free RL.

Related to the first principle of relying on the data is the second key difference of reinforcement learning methods from DP methods. Because data is always noisy, reinforcement learning cannot aim at an exact solution as the standard DP methods, but rather aim at some good approximate, rather than exact, solutions. Clearly, this does not prevent an exploration of the behavior of RL solutions when the number of data points becomes infinite. If we knew an exact model of the world, we could reconstruct it exactly in this limit. Therefore, theoretically sound (rather than purely empirically driven) RL algorithms should demonstrate convergence to known solutions in this asymptotic limit. In particular, if we deal with a sufficiently low-dimensional system, such solutions can be independently calculated using the standard DP methods. This can be used for testing and benchmarking RL algorithms, as will be discussed in more details in later sections of this book.

Finally, the last key difference of reinforcement learning methods from DP is that they do not seek the best solution, they simply seek a “sufficiently good” solution. The main motivation for such a paradigm change is the “curse of dimensionality” that was mentioned above. DP methods for finite MDPs operate with tabular representations of value functions, rewards, and transition probabilities. Memory requirements and speed constraints make this approach infeasible for high-dimensional discrete or continuous state-action spaces. Therefore, when working with such problems, reinforcement learning approaches rely on function approximation for quantities of interest such as value functions or action policies. Reinforcement learning algorithms with function approximation will be discussed later in this section, after we introduce tabulated versions of these algorithms for finite MDPs with sufficiently low dimensionality of discrete-valued state and action spaces.

The purpose of this section is to introduce some of the most popular RL algorithms for both finite and continuous-state MDP problems. We will start with methods developed for finite MDPs, and then later show how they can be extended to continuous state-action spaces using function approximation approaches.

? Multiple Choice Question 1

Select all the following correct statements:

- a. Unlike DP, RL needs to know rewards and transition probability functions.
 - b. Unlike DP, RL does not need to know reward and transition probability functions, as it relies on samples.
 - c. The information set \mathcal{F}_t for RL includes a triplet $(X_t^{(n)}, a_t^{(n)}, X_{t+1}^{(n)})$ for each step.
 - d. The information set \mathcal{F}_t for RL includes a tuple $(X_t^{(n)}, a_t^{(n)}, R_t^{(n)}, X_{t+1}^{(n)})$ for each step.
-

5.1 Monte Carlo Methods

Monte Carlo methods, as other methods of reinforcement learning, do not assume complete knowledge of the environment, nor do they rely on any model of the environment. Instead, Monte Carlo methods rely on *experience*, that is samples of states, actions, and rewards. When working with real data, this amounts to learning without any prior knowledge of the environment. Experience can also be simulated. In this case, Monte Carlo methods provide a simulation-based approach to solving MDP problems. Recall that the DP approach requires knowledge of exact transition probabilities to perform iteration steps in policy iteration or value iteration algorithms. With reinforcement learning Monte Carlo methods, only samples from these distributions are needed, but not their explicit form.

Monte Carlo methods are normally restricted to episodic task with a finite planning horizon $T < \infty$. Rather than relying on Bellman equations, they operate directly with the definition of the action-value function

$$Q_t^\pi(s, a) = \mathbb{E}_t^\pi \left[\sum_{i=0}^{T-1} \gamma^{t-i} R(S_{t+i}, a_{t+i}, S_{t+i+1}) \middle| S_t = s, a_t = a \right] = \mathbb{E}_t^\pi [G_t | S_t = s, a_t = a], \quad (9.33)$$

where G_t is the total return, see Eq. (9.11). If we have access to data consistent of N set of T -step trajectories each producing return $G_t^{(n)}$, then we could estimate the action-value function at the state-action values (s, a) using the empirical mean:

$$Q_t^\pi(s, a) \simeq \frac{1}{N} \sum_{n=1}^N [Q_t^{(n)} | S_t = s, a_t = a]. \quad (9.34)$$

Note that for each trajectory, a complete T -step trajectory should be observed, so that its return can be observed and used to update the action-value function.

It is worth clarifying the meaning of index π in this relation. With the Monte Carlo estimation of Eq. (9.34), π should be understood as a policy that was applied when collecting the data. This means that this Monte Carlo method is an *on-policy* algorithm. On-policy algorithms are only able to learn an optimal policy from samples if these samples themselves are produced using the optimal policy.

Conversely, *off-policy* algorithms are able to learn an optimal policy from data generated using other, sub-optimal policies. Off-policy Monte Carlo methods will not be addressed here, and the interested reader is referred to Sutton and Barto (2018) for more details on this topic.

As indicated by Eq. (9.33), the action-value process should be calculated separately for each combination of a state and action in a finite MDP assumed here. The number of such combinations will be $|\mathcal{S}| \cdot |\mathcal{A}|$. Respectively, for each combination of s and a from this set, we should only select those trajectories that encounter such a combination, and only include returns from these trajectories in the sum in Eq. (9.34). For every combination of (s, a) , the empirical estimate (9.34) asymptotically converges to the exact answer in the limit $N \rightarrow \infty$. Also note that these estimates are independent for different values of (s, a) . This could be useful, as it enables a trivial parallelization of the calculation. On the other hand, independence of estimates for different pairs (s, a) means that this algorithm does not *bootstrap*, i.e. it does not use previous or related evaluations to estimate the action-value function at node (s, a) . Such a method may miss some regularities observed or expected in true solutions (e.g., smoothness of the value function with respect to its arguments), and therefore may produce some spurious jumps in estimated state-value functions produced due to noise in the data.

Beyond empirical estimation of the action-value function as in Eq. (9.34) or the state-value function, Monte Carlo methods can also be used to find the optimal control, provided the Monte Carlo RL agent has access to a real-world or simulated environment. To this end, the agent should produce trajectories using different trial

policies. For each policy π , a number of N trajectories are sampled using this policy. The action-value function is estimated using an empirical mean as in Eq. (9.34). After this, one follows with the policy improvement step which coincides with the greedy update of the policy iteration method: $\pi'(s) = \arg \max_a Q^\pi(s, a)$. The new policy is used to sample a new set of trajectories, and the process runs until convergence or for a fixed number of steps.

Note that generating new trajectories corresponding to newly improved policies may not always be feasible. For example, an agent may only have access to one fixed set of trajectories obtained using a certain fixed policy. In such cases, one may resort to *importance sampling* techniques which use trajectories obtained under different policies to estimate the return under a given policy. This is achieved by re-weighting observed trajectories by likelihood ratio factors obtained as ratios of probabilities of observing given reward under the trial policy π' and the policy π used in the data collection stage.

Instead of updating the action-value function (or the state-value function) simultaneously after all N trajectories are sampled, which is a *batch mode* evaluation, we could convert the problem into an online learning problem where updates occur after observing each individual trajectory according to the following rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha [G_t(s, a) - Q(s, a)], \quad (9.35)$$

where $0 < \alpha < 1$ is a step-size parameter usually referred to as the “learning rate.” It can be shown that such iterative update converges to true empirical and theoretical averages in the limit $N \rightarrow \infty$. Yet this update is not entirely real time, as it requires finishing each T -step trajectory, until its total return G_t can be used in the update (9.35). This may be inefficient, especially if multiple trajectory generations and evaluations are required as a part of policy optimization. As we will show below, there are other methods of learning that are free of this drawback.

5.2 Policy-Based Learning

In value-based RL, the optimal policy is obtained from an optimal value function, and thus is not modeled separately. *Policy-based* reinforcement learning takes a different approach, and directly models policy. Unlike the value-based RL where we considered *deterministic* policies, policy-based RL operates with *stochastic* policies $\pi_\theta(a|s)$ that define probability distributions over a set of possible actions $a \in \mathcal{A}$, where θ defines parameters of this distribution.

Recall that deterministic policies can be considered special cases of stochastic policies where a distribution of possible actions degenerates into a Dirac delta-function concentrated in a single action prescribed by the policy: $\pi_\theta(a|s) = \delta(a - a_*(s, \theta))$ where $a_*(s, \theta)$ is a fixed map from states to actions, parameterized by θ . With either deterministic or stochastic policies, learning is performed by tuning the free parameters θ to maximize the total expected reward.

Policy-based methods are based on a simple relation commonly known as the “log-likelihood trick” which is obtained by computing the derivative of an expectation $J(\theta) = \mathbb{E}_{\pi_\theta(a)} [G(a)]$. Here function $G(a)$ can be arbitrary, but to connect to reinforcement learning, we will generally mean that $G(a)$ stands for the expectation of the random return (9.11), which we write here as $G(a)$ to emphasize its dependence on the actions taken. The gradient of the expectation with respect to parameters θ can be computed as follows:

$$\nabla_\theta J(\theta) = \int G(a) \nabla_\theta \pi_\theta(a) dz = \int G(a) \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} \pi_\theta(a) da = \mathbb{E}_{\pi_\theta(a)} [G(a) \nabla_\theta \log \pi_\theta(a)]. \quad (9.36)$$

This shows that the gradient of J with respect to θ is the expected value of the function $G(a) \nabla_\theta \log \pi_\theta(a)$. Therefore, if we can sample from the distribution $\pi_\theta(a)$, we can compute this function and have an unbiased estimate of the gradient of $G(a)$ by sampling. This is the reason the relation (9.36) is called the “log-likelihood trick”: it allows one to estimate the gradient of the functional $J(\theta)$ by sampling or simulation.

The log-likelihood trick underlies the simplest policy search algorithm called REINFORCE. The algorithm starts with some initial values of parameters θ_0 and the iteration counter is $k = 0$. Using a size-step hyperparameter α , the update of θ_k amounts to first sampling $a_k \sim p_{\pi_k}(a)$, and then updating the vector of parameters using the incremental version of Eq. (9.36)

$$\theta_{k+1} = \theta_k + \alpha_k G(a_k) \nabla_\theta \log \pi_{\theta_k}(a_k). \quad (9.37)$$

Here α is a learning rate parameter defining the speed of updates along the negative of the gradient of $G(a)$. The algorithm continues until convergence, or for a fixed number of steps. As one can see, this algorithm is very simple to implement as long as sampling from distribution $\pi_\theta(a)$ is easy. On the other hand, Eq. (9.37) is a version of stochastic gradient descent which can be noisy and thus produce high variance estimators.

The REINFORCE algorithm (9.37) is a pure policy search method that does not use any value function. A more sophisticated version of learning can be obtained where we simultaneously model both the policy and the action-value functions. Such methods are called *actor-critic* methods, where “actor” is an algorithm that generates a policy from a family $\pi_\theta(a|x)$, and “critic” evaluates the results of applying the policy, expressing it in terms of a state-value or action-value function. Following such terminology, the REINFORCE algorithm could be considered an “actor-only” algorithm, while SARSA or Q-learning, to be presented below, could be viewed as “critic-only” methods.

One advantage of policy-based algorithms is that they admit very flexible parameterizations of action policies, which can also work for continuous action spaces. One popular and quite general type of action policies is the so-called *softmax in actions* policy

$$\pi_\theta(a|s) = \frac{e^{h(s,a,\theta)}}{\sum_{a'} e^{h(s,a',\theta)}}. \quad (9.38)$$

Functions $h(s, a, \theta)$ in this expression would be interpreted as action preferences. They could be taken linear functions in parameters θ , for example

$$h(s, a, \theta) = \theta^T U(s, a), \quad (9.39)$$

where $U(s, a)$ would be a vector of features constructed on the product space $\mathcal{S} \times \mathcal{A}$. Models of this kind are known as *linear architecture* models. Alternatively, preference functions $h(s, a, \theta)$ could be modeled non-parametrically using neural networks (or some other universal approximators such as decision trees). Parameters θ in this case would be weights of such neural network. Indeed many implementations of actor-critic algorithms involve using two separate neural networks that serve as general function approximations for the action policy and value functions, respectively. More on actor-critic algorithms can be found in Sutton and Barto (2018), Szepesvari (2010).

5.3 Temporal Difference Learning

We have seen that Monte Carlo methods must wait until the end of each episode to determine the increment of the action-value function update (9.35). *Temporal difference* (TD) methods perform updates differently, by waiting only until the next time step, and incrementing the value function at each time step. TD methods can be used for both policy evaluation and policy improvements. Here we focus on how they can be used to evaluate a given policy π by computing a state-value function V_t^π .

How it can be done can be seen from the Bellman equation (9.14) which we repeat here for convenience:

$$V_t^\pi(s) = \mathbb{E}_t^\pi [R_t(s, a, s') + \gamma V_{t+1}^\pi(s')]. \quad (9.40)$$

As we discussed above, this equation can be converted into an update equation, if the state-value function from a previous iteration is used to evaluate the right-hand side to define the update rule. This idea was used in value iteration and policy iteration algorithms of DP. TD methods use the same idea, and add to this an estimation of the expectation entering Eq. (9.40) from a single observation—which is the observation obtained at the next time step. Without relying on a theoretical model of the world which, similarly to the DP approach, would calculate the expectation in Eq. (9.40) exactly, TD methods rely on a simplest possible estimation of this expectation, essentially by computing an empirical mean from a single observation!

Clearly, relying on a single observation to estimate an empirical mean can lead to very volatile updates, but this is the price one should be prepared to pay for a truly online method. On the other hand, it is extremely fast. Even if it might bring only a marginal improvement (on average) for maximization of a value function, it might produce a workable and efficient algorithm, because such updates may be repeated many times and at a low cost during steps of policy improvement.

A TD method, when applied to the state-value function $V_t^\pi(s)$, takes a mismatch between the right-hand side of Eq. (9.40) (estimated with a single observation) and its left-hand side as a measure of an error δ_t , also called the TD error:

$$\delta_t = R_t(s, a, s') + \gamma V_{t+1}(s') - V_t(s). \quad (9.41)$$

Note as we deal here with updating the state-value function for a fixed policy π , we omitted explicit upper indices π in this relation. This error defines the rule of update of the state-value function at node s :

$$V_t(s) \leftarrow V(s) + \alpha [R_t(s, a, s') + \gamma V_{t+1}(s') - V_t(s)], \quad (9.42)$$

where α is a learning rate. Note that the learning rate should not be a constant, but rather can vary with the number of iterations. In fact, as we will discuss shortly, a certain decay schedule for the learning rate $\alpha = \alpha_t$ should be implemented to guarantee convergence where the number of updates goes to infinity.

Note that the TD error δ_t is not actually available at time t as it depends on the next-step value s' , and is therefore only available at time $t + 1$. The update (9.42) relies only on the information from the next step, and is therefore often referred to as the one-step TD update, also known as the TD(0) rule. It is helpful to compare this with Eq. (9.35) that was obtained for a Monte Carlo (MC) method. Note the Eq. (9.35) requires observing a whole episode in order to compute the full trajectory return G_t . Rewards observed sequentially do not produce updates of a value function until a trajectory is completed. Therefore, the MC method cannot be used in an online setting.

On the other hand, the TD update rule (9.42) enables updates of the state-value function after each individual observation, and therefore can be used as an online algorithm that *bootstraps* by combining a reward signal observed in a current step with an estimation of a next-period value function, where both values are estimated from a sample. TD methods thus combine the bootstrapping properties of DP with a sample-based approach of Monte Carlo methods. That is, updates in TD methods are based on samples, while in DP they are based on computing expectations of next-period value functions using a specific model of the environment. For any fixed policy π , the TD(0) rule (9.42) can be proved to converge to a true state-value function if the learning rate α slowly decreases with the number of iterations. Such proofs of convergence hold for both finite MDPs and for MDPs with continuous

state-action spaces—with the latter case only established with a linear function approximation, but not with more general non-linear function approximations.⁵

The ability of TD learning to produce updates after each observation turns out to be very important in many practical applications. Some RL tasks involve long episodes, and some RL problems such as continuous learning do not have any unambiguous definition of finite-length episodes. Whether an episodic learning appears natural or *ad hoc*, delaying learning until the end of each episode can slow it down and produce inefficient algorithms. Because each update of model parameters requires re-running of all episodes, Monte Carlo methods become progressively more inefficient in comparison to TD methods with increased model complexity.

There exist several versions of TD learning. In particular, instead of applying it to learning a state-value function $V_t(s)$, we could use a similar approach to update the action-value function $Q_t(s, a)$. Furthermore, for both types of TD learning, instead of one-step updates such as the TD(0) rule (9.42), one could use multi-step updates, leading to more general TD(λ) methods and n -step TD methods. We refer the reader to Sutton and Barto (2018) for a discussion on these algorithms, while focusing in the next section on one-step TD learning methods for an action-value function.

5.4 SARSA and Q-Learning

We now arrive at arguably the most important material in this chapter. In applying TD methods to learn an action-value function $Q(s, a)$ instead of a state-value function $V(s)$, one should differentiate between *on-policy* and *off-policy* algorithms. Recall that *on-policy* algorithms assume that policy used to produce a dataset used for learning is an optimal policy, and the task is therefore to learn the optimal policy function from the data. In contrast, *off-policy* algorithms assume that the policy used in a particular dataset may not necessarily be an optimal policy, but can be sub-optimal or even purely random. The purpose of off-policy algorithms is to find an optimal policy when data is collected under a different policy. This task is in general more difficult than the first case of on-policy learning which can be viewed as a direct inference problem of fitting a function (a policy function, in this case) to observed data.

For both on-policy and off-policy learning with TD methods, the starting point is the Bellman optimality equation (9.20) that we repeat here

$$Q_t^\star(s, a) = \mathbb{E}_t^\star \left[R_t(s, a, s') + \gamma \max_{a'} Q_{t+1}^\star(s', a') \right]. \quad (9.43)$$

⁵We will discuss function approximations below, after we present TD algorithms in a tabulated setting that is appropriate for finite MDPs with a sufficiently low number of possible states and actions.

The idea of TD methods for the action-value function is the same as before, to estimate the right-hand side of Eq. (9.43) from observations, and then use a mismatch between the right- and left-hand sides of this equation to define the rule of an update. However, details of such procedure depend on whether we use on-policy or off-policy learning.

Consider first the case of on-policy learning. If we know that data was collected under an optimal policy, the max operator in Eq. (9.43) becomes redundant, as observed actions should in this case correspond to maximum of a value function. Similar to the TD method for the state-value function, we replace the expectation in (9.43) by its estimation based on a single observation. The update in this case becomes

$$Q_t(s, a) \leftarrow Q_t(s, a) + \alpha [R_t(s, a, s') + \gamma Q_{t+1}(s', a') - Q_t(s, a)]. \quad (9.44)$$

This on-policy algorithm is known as SARSA, to emphasize that it uses a quintuple (s, a, r, s', a') to make an update. The TD error for this case is

$$\delta_t = R_t(s, a, s') + \gamma Q_{t+1}(s', a') - Q_t(s, a). \quad (9.45)$$

Convergence of the SARSA algorithm depends on a policy used to generate data. If the policy converges to a greedy policy in the limit of an infinite number of steps, SARSA converges to the true policy and action-value functions in the limit when each state-action pair is visited an infinite number of times.

➤ SARSA vs Q-Learning

- *SARSA* is an *On-Policy method*, which means that it computes the Q-value according to a certain policy and then the agent follows that policy.
- *Q-learning* is an *Off-Policy method*. It consists of computing the Q-value according to a greedy policy, but the agent does not necessarily follow the greedy policy.

Now consider the case of off-policy learning. In this case, the data available for learning is collected using some sub-optimal, or possibly even purely random policy. Can we still learn from such data? The answer to this question is in the affirmative - we simply replace the expectation in (9.43) by its estimate obtained from a single observation, as in SARSA, but keeping this time the max operator over next-step actions a' :

$$Q_t(s, a) \leftarrow Q_t(s, a) + \alpha \left[R_t(s, a, s') + \gamma \max_{a'} Q_{t+1}(s', a') - Q_t(s, a) \right]. \quad (9.46)$$

This is known as *Q-learning*. It was proposed by Watkins in 1989, and since then has become one of the most popular approaches in reinforcement learning. Q-learning is provably convergent for finite MDPs when the learning rate α slowly decays with the number of iterations, in the limit when each state-action pair is visited an infinite number of times. Extensions of a tabulated-form Q-learning (9.46) for finite MDPs to systems with a continuous state space will be presented in the following sections. Q-learning is thus a TD(0) learning applied to an action-value function.

Note the key difference between SARSA and Q-learning in an online setting when an agent has to choose actions during learning. In SARSA, we use the same policy (e.g., an ε -greedy policy, see Exercise 9.8) to generate both the current action a and the next action a' . In contrast to that, in Q-learning the next action a' is a greedy action that maximizes the action-value function $Q_{t+1}(s', a')$ at the next time moment. It is exactly the choice of a greedy next action a' that makes Q-learning an off-policy algorithm that can learn an optimal policy from different and sub-optimal execution policies.

The reason Q-learning works as an off-policy method is that the TD(0) rule (9.46) does not depend on the policy used to obtain data for training. Such dependence enters the TD rule only indirectly, via an assumption that this policy should be such that each state-action pair is encountered in the data many times—in fact, an infinite number of times asymptotically, when the number of observations goes to infinity. The TD rule (9.46) does not try to answer the question *how* the observed values of such pairs are computed. Instead, it directly uses these observed values to make updates in values of the action-value function.

It is its ability to learn from off-policy data that makes Q-learning particularly attractive for many tasks in reinforcement learning. In particular, in batch reinforcement learning, an agent should learn from some data previously produced by some other agent. Assuming that the agent that produced the historical data was acting optimally may in many cases be too stringent or unrealistic an assumption. Moreover, in certain cases the previous agent might have been acting optimally, but an environment could change due to some trend (drift) effects. Even though a policy used in the data could be optimal for previous periods, due to the drift, this becomes off-policy learning. In short, it appears there are more examples of off-policy learning in real-world applications than of on-policy learning.

The ability to use off-policy data does not come without a price which is related to the presence of the max operator in Eq. (9.46). This operator in fact provides a mechanism for comparison between different policies during learning. In particular, Eq. (9.46) implies that one cannot learn an optimal policy from a single observed transition $(s, a) \rightarrow (r, s', a')$ and nothing else, as the chosen next action a' may not necessarily be an optimal action that maximizes $Q_{t+1}^*(s', a')$, as required in the Q-learning update rule (9.46).

This suggests that online Q-learning could maintain a tabulated representation of the action-value function $Q(s, a)$ for all previously visited pairs (s, a) , and use it in order to estimate the $\max_{a'} Q_{t+1}^*(s', a')$ term using both the past data and a newly observed transition. Such a method could be viewed as an incremental version of batch learning where a batch dataset is continuously updated by

adding new observations, and possibly remove observations that are too old and possibly correspond to very sub-optimal policies. This approach is called *experience replay* in the reinforcement learning literature. The same procedure of adding one observation (or a few of them) at the time can also be used in a pure batch-mode Q-learning as a computational method that allows one to make updates as processing trajectories stored in the datafile. The difference between online Q-learning with experience replay and a pure batch-mode Q-learning therefore amounts to different rules of updating the batch file. In batch-mode Q-learning, it stays the same during learning, but could also be built in increments of one or a few observations to speed up the learning process. In online Q-learning, the experience replay buffer is continuously updated by adding new observations, and removing distant ones to keep the buffer size fixed.

Example 9.4 Financial Cliff Walking with SARSA and Q-Learning

The “financial cliff walking” example introduced earlier in this chapter can serve as a simple test case for SARSA and Q-learning for a finite MDP. We assume $N = 4$ values of possible funds in the account, and assume $T = 12$ time steps. All combinations of state and time can then be represented as a two-dimensional grid of size $N \times T = 4 \times 12$. A time-dependent action-value function $Q_t(s_t, a_t)$ with three possible actions $a_t = \{a_+, a_-, a_0\}$ can then be stored as a rank-three tensor of dimension $4 \times 12 \times 3$.

To facilitate exploration required in online applications of RL, we can use a ε -greedy policy. The ε -greedy policy is a simple stochastic policy where the agent takes an action that maximizes the action-value function with probability $1 - \varepsilon$, and takes a purely random action with probability ε . The ε -greedy policy is used to produce both actions a, a' in the SARSA update (9.44), while with Q-learning it is only used to pick the action at the current step. A comparison of the optimal policies is given in Table 9.1. For sufficiently small α and under tapering of ε (see Fig. 9.3), both methods are shown by Fig. 9.4 to converge to the same cumulative reward. This example is implemented in the financial cliff walking with Q-learning notebook. See Appendix “Python Notebooks” for further details.

5.5 Stochastic Approximations and Batch-Mode Q-learning

A more systematic view of TD methods is given by their interpretation as stochastic approximations to solve Bellman equations. As we will show in this section, such a view both helps to better understand the meaning of TD update rules presented above, as well as to extend them to learning using batches of observations in each step, instead of taking all individual observations one by one.

Table 9.1 The optimal policy for the financial cliff walking problem using (top) SARSA (S) and (below) Q-learning (Q). The row indices denote the balance and the column indices denote the time period. Note that the two optimal policies are almost identical. Starting with a zero balance, both optimal policies will almost surely result in the agent following the same shortest path, with a balance of 1, until the final time period

S	0	1	2	3	4	5	6	7	8	9	10	11
3	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
2	Z	Z	Z	Z	Z	Z	Z	Z	Z	D	Z	Z
1	Z	Z	D	Z								
0	U	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	G
Q	0	1	2	3	4	5	6	7	8	9	10	11
3	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
2	Z	Z	Z	Z	Z	Z	Z	Z	Z	D	D	Z
1	Z	Z	D	Z								
0	U	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	G

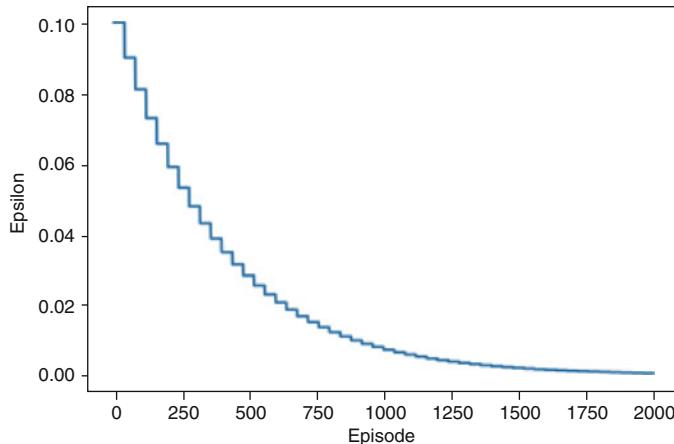


Fig. 9.3 This figure illustrates how ϵ is tapered in the financial cliff walking problem with increasing episodes so that Q-learning and SARSA converge to the same optimal policy and cumulative reward as shown in Table 9.1 and Fig. 9.4

When the model on an environment is unknown, we try to *approximately* solve the Bellman optimality equation (9.20) by replacing expectations entering this equation by their empirical averages. Stochastic approximations such as the Robbins–Monro algorithm (Robbins and Monro 1951) take this idea one step further, and estimate the mean without directly summing the samples.

We can illustrate the idea behind this method using a simple example of estimation of a mean value $\frac{1}{K} \sum_{k=1}^K x_k$ of a sequence of observations x_k with $k = 1, \dots, K$. Instead of waiting for all K observations, we can add them one by one, and iteratively update the running estimation of the mean \hat{x}_k , where k is the number of iteration, or the number of data points in a dataset:

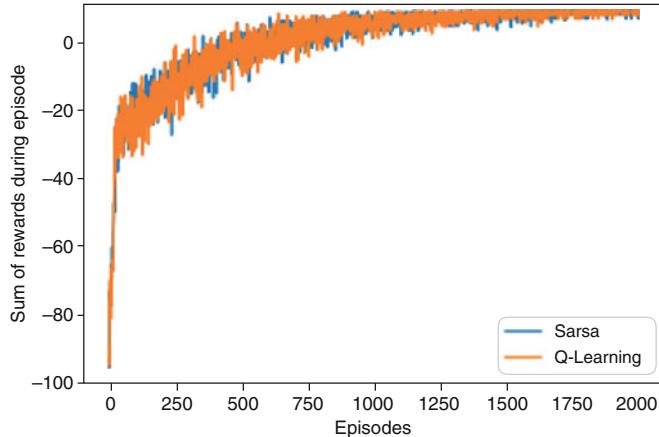


Fig. 9.4 Q-learning and SARSA are observed to converge to almost the same optimal policy and cumulative reward in the financial cliff walking problem under the ϵ -tapering schedule shown in Fig. 9.3. Note that the cumulative rewards are averaged over twenty simulations

$$\hat{x}_{k+1} = (1 - \alpha_k)\hat{x}_k + \alpha_k x_k, \quad (9.47)$$

and where $\alpha_k < 1$ denotes the step size (learning rate) at step k , that should satisfy the following conditions:

$$\lim_{K \rightarrow \infty} \sum_{k=1}^K \alpha_k = \infty, \quad \lim_{K \rightarrow \infty} \sum_{k=1}^K (\alpha_k)^2 < \infty. \quad (9.48)$$

Robbins and Monro have shown that under these constraints, an iterative method of computing the mean (9.47) converges to a true mean with probability one (Robbins and Monro 1951). In general, the optimal choice of a (step-dependent) learning rate α_k is not universal but specific to a problem, and may require some experimentation.

Q-learning presented in Eq. (9.46) can now be understood as the Robbins–Monro stochastic approximation (9.47) to estimate the unknown expectation in Eq. (9.43) as a current estimate $Q_t^{(k)}(s, a)$ corrected by a current observation $R_t(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}^{(k)}(s', a')$:

$$Q_t^{(k+1)}(s, a) = (1 - \alpha_k)Q_t^{(k)}(s, a) + \alpha_k \left[R_t(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}^{(k)}(s', a') \right]. \quad (9.49)$$

The single-observation Q -update in Eq. (9.49) corresponds to a pure online version of the Robbins–Monro algorithm. Alternatively, stochastic approximations can be employed in an off-line manner, by using a *chunk* of data, instead of a single data point, to iteratively update model parameters. Such approaches are useful when working with large datasets, and are frequently used in machine learning, e.g. for a mini-batch stochastic gradient descent method, as a way to more efficiently train a

model by feeding it mini-batches of data. In addition, batch versions of stochastic approximation methods are widely used when doing reinforcement learning in continuous state-action spaces, which is a topic we discuss next.

? Multiple Choice Question 2

Select all the following correct statements:

- a. Q-learning is obtained by using the Robbins–Monro stochastic approximation to estimate the $\max(\cdot)$ term in the Bellman optimality equation.
 - b. Q-learning is obtained by using the Robbins–Monro stochastic approximation to estimate the unknown expectation in the Bellman optimality equation.
 - c. The optimal Q-function in Q-learning is obtained when an optimal learning rate $\alpha_k = \frac{\alpha}{k}$ where $\alpha \simeq 1/137$ is used for learning.
 - d. The optimal Q-function is learned in Q-learning iteratively, where each step (Q-iteration) implements one iteration of the Robbins–Monro algorithm.
-

Example 9.5 Optimal Stock Execution with SARSA and Q-Learning

A setting that is very similar to the “financial cliff walking” example introduced earlier in this chapter can serve to develop a toy MDP model for optimal stock execution.

Assume that the broker has to sell N blocks of shares with n shares in each block, e.g. we can have $N = 10$, $n = 1000$. The state of the inventory at time t is then given by the variable X_t taking values in a set \mathcal{X} with $N = 10$ states $\mathcal{X}^{(n)}$, so that the start point at $t = 0$ is $X_0 = X^{(N-1)}$ and the target state is $X_T = X^{(0)} = 0$. In each step, the agent has four possible actions $a_t = a^{(i)}$ that measure the number of blocks of shares sold at time t where $a^{(0)} = 0$ stands for no action, and $a^{(i)} = i$ with $i = 1, \dots, 3$ is the number of blocks sold. The update equation is

$$X_{t+1} = (X_t - a_t)_+. \quad (9.50)$$

Trades influence the stock price dynamics through a linear market impact

$$S_{t+1} = S_t e^{(1-\nu a_t)} + \sigma S_t Z_t, \quad (9.51)$$

where ν is a market friction parameter. To map onto a finite MDP problem, a range of possible stock prices \mathcal{S} can be discretized to M values, e.g. $M = 12$. The state space of the problem is given by a direct product of states $\mathcal{X} \times \mathcal{S}$ of dimension $N \times M = 10 \cdot 12 = 120$. The dimension of the extended space including the time is then $120 \cdot 10 = 1200$.

(continued)

Example 9.5 (continued)

The payoff of selling a_t blocks of shares when the stock price is S_t is $na_t S_t$. A risk-adjusted payoff adds a penalty on variance of the remaining inventory price at the next step $t + 1$: $r_t = na_t S_t - \lambda n \text{Var}[S_{t+1} X_{t+1}]$. All combinations of state and time can then be represented as a three-dimensional grid of size $N \times M \times T = 10 \cdot 12 \cdot 10$. A time-dependent action-value function $Q_t(s_t, a_t)$ with four possible actions $a_t = \{a_0, a_1, a_2, a_3\}$ can then be stored as a rank-four tensor of dimension $10 \times 12 \times 10 \times 4$.

We can now apply SARSA or Q-learning to learn optimal stock execution in such a simplified setting. For exploration needed for online learning, one can use a ε -greedy policy. At each time step, a time-dependent optimal policy is therefore found with 10×12 (for inventory and stock price level) states and four possible actions $a_t = \{a_0, a_1, a_2, a_3\}$ can be viewed as a 10×12 matrix as shown, for the second time step, in Table 9.2. This example is implemented in the market impact problem with Q-learning notebook. See Appendix “Python Notebooks” for further details (Fig. 9.5).

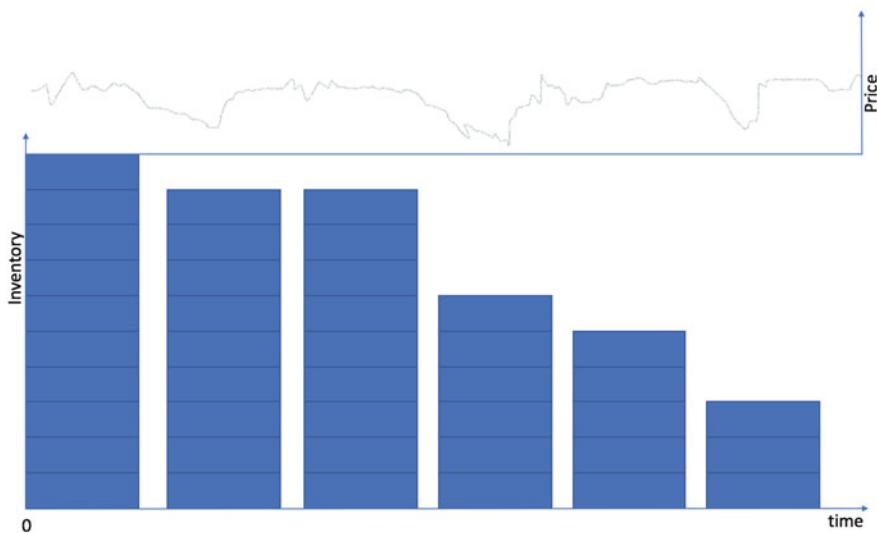


Fig. 9.5 The optimal execution problem: how to break up large market orders into smaller orders with lower market impact? In the finite MDP formulation, the state space is the inventory, shown by the number of blocks, stock price, and time. In this illustration, the agent decides whether to sell $\{0, 1, 2, 3\}$ blocks at each time step. The problem is whether to retain inventory, thus increasing market risk but reduces the market impact, or quickly sell inventory to reduce exposure but increase the market impact

Table 9.2 The optimal policy, at time step $t = 2$, for the trade execution problem using (left) SARSA and (right) Q-learning. The rows denote the inventory level and the columns denote the stock price level. Each element denotes an action to sell {0, 1, 2, 3} blocks of shares.

Example 9.6 Electronic Market Making with SARSA and Q-Learning

We can build on the previous two examples by considering the problem of high-frequency market making. Unlike the previous example, we shall learn a time-independent optimal policy.

Assume that a market maker seeks to capture the bid–ask spread by placing one lot best bid and ask limit orders. They are required to strictly keep their inventory between -1 and 1 . The problem is when to optimally bid to buy (“ b ”), bid to sell (“ s ”), or hold (“ h ”), each time there is a limit order book update. For example, sometimes it may be more advantageous to quote a bid to close out a short position if it will almost surely yield an instantaneous net reward, other times it may be better to wait and capture a larger spread.

In this toy example, the agent uses the liquidity imbalance in the top of the order book as a proxy for price movement and, hence, fill probabilities. The example does not use market orders, knowledge of queue positions, cancelations, and limit order placement at different levels of the ladder. These are left to later material and exercises.

A simple illustration of the market making problem is shown in Fig. 9.6. At each non-uniform time update, t , the market feed provides best prices and depths $\{p_t^a, p_t^b, q_t^a, q_t^b\}$. The state space is the product of the inventory, $X_t \in \{-1, 0, 1\}$, and gridded liquidity ratio $\hat{R}_t = \lfloor \frac{q_t^a}{q_t^a + q_t^b} N \rfloor \in [0, 1]$, where N is the number of grid points and q_t^a and q_t^b are the depths of the best ask and bid. $\hat{R}_t \rightarrow 0$ is the regime where the mid-price will go up and an ask is filled. Conversely for $\hat{R}_t \rightarrow 1$. The dimension of the state space is chosen to be $3 \cdot 5 = 15$.

A bid is filled with probability $\epsilon_t := \hat{R}_t$ and an ask is filled with probability $1 - \epsilon_t$. The rewards are chosen to be the expected total P&L. If a bid is filled to close out a short holding, then the expected reward $r_t = -\epsilon_t(\Delta p_t + c)$, where Δp_t is the difference between the exit and entry price and c is the transaction cost. For example, if the agent entered a short position at time $s < t$ with a filled ask at $p_s^a = 100$ and closed out the position with a filled bid at $p_t^b = 99$, then $\Delta p_t = 1$. The agent is penalized for quoting an ask or bid when the position is already short or long, respectively.

As with previous examples, we apply SARSA or Q-learning to optimize market making. For exploration needed for online learning, one can use a ϵ -greedy policy. A comparison of the optimal policies is given in Table 9.3. For sufficiently large number of iterations in each episode and under tapering of ϵ , both methods are observed to converge to the same cumulative reward in Fig. 9.7. This example is implemented in the electronic market making with Q-learning notebook. See Appendix “Python Notebooks” for further details.

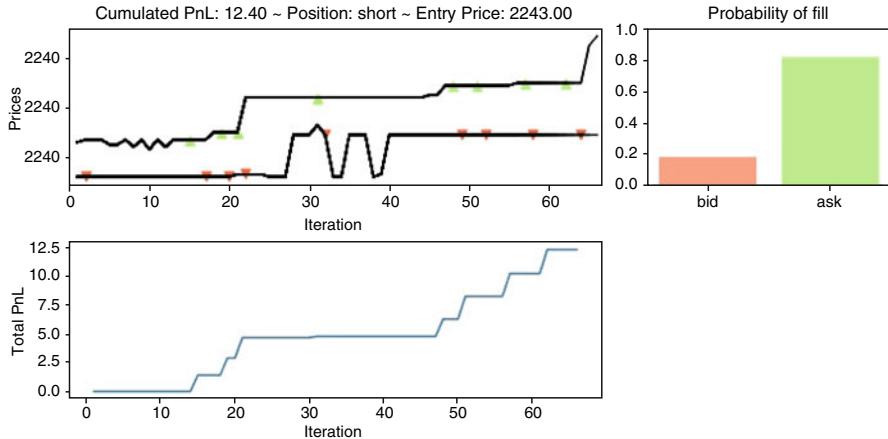


Fig. 9.6 The market making problem requires the placement of bid and ask quotes to maximize P&L while maintaining a position within limits. For each limit order book update, the agent must anticipate which quotes shall be filled to capture the bid–ask spread. Transaction costs, less than a tick, are imposed to penalize trading. This has the net effect of rewarding trades which capture at least a tick. A simple model is introduced to determine the fill probabilities and the state space is the product of the position and gridded fill probabilities

Table 9.3 The optimal policy for the market making problem using (top) SARSA (S) and (below) Q-learning (Q). The row indices denote the position and the column indices denote the predicted ask fill probability buckets. Note that the two optimal policies are almost identical

S	0–0.1	0.1–0.2	0.2–0.3	0.3–0.4	0.4–0.5	0.5–0.6	0.6–0.7	0.7–0.8	0.8–0.9	0.9–1.0
Flat	b	b	b	b	b	b	s	s	s	s
Short	b	b	b	b	b	b	b	b	b	h
Long	h	s	s	s	s	s	s	s	s	s
Q	0–0.1	0.1–0.2	0.2–0.3	0.3–0.4	0.4–0.5	0.5–0.6	0.6–0.7	0.7–0.8	0.8–0.9	0.9–1.0
Flat	b	b	b	b	b	b	s	b	b	s
Short	b	b	b	b	b	b	b	b	b	b
Long	s	s	s	s	s	s	s	s	s	s

5.6 Q-learning in a Continuous Space: Function Approximation

Our previous presentation of reinforcement learning algorithms assumed a setting of a finite MDP model with discrete state and action spaces. For this case, all combinations of state-action pairs are enumerable. Therefore, the action-value and state-value functions can be maintained in a tabulated form, while TD methods such as Q-learning (9.49) can be used to compute the action-value function values at these nodes in an iterative manner.

While such methods are simple and can be proven to converge in a tabulated setting, they also quickly run into their limitations for many interesting real-world

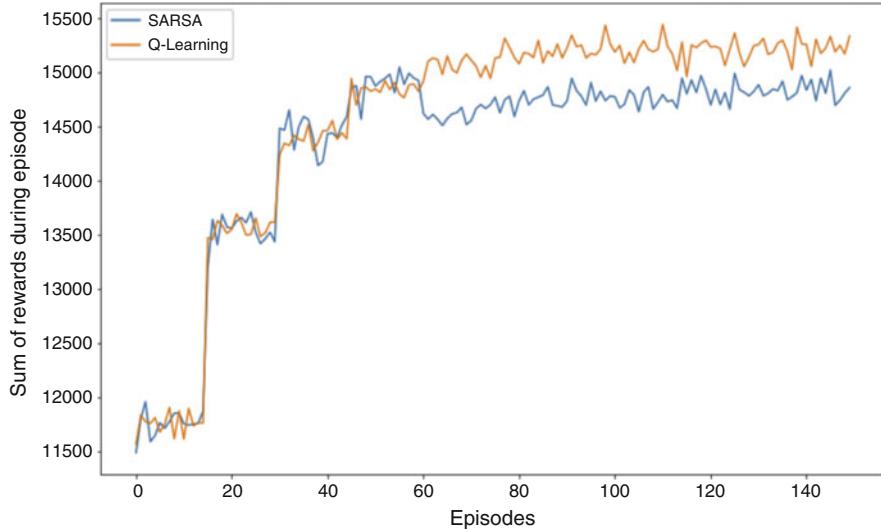


Fig. 9.7 Q-learning and SARSA are observed to converge to almost the same optimal policy and cumulative reward in the market making problem

problems. The latter are often high-dimensional in discrete or continuous state-action spaces. If we use a straightforward discretization of each continuous state and/or action variable, and then enumerate all possible combinations of states and actions, we end up with an exponentially large number of state-action pairs. Even storing such data can pose major challenges with memory requirements, not to speak about an exponential slow-down in a cost of computations, where all such pairs essentially become parameters of a very high-dimensional optimization problem. This calls for approaches based on function approximation methods where functions in (high-dimensional) discrete or continuous spaces are represented in terms of a relatively small number of freely adjustable parameters.

To motivate linear function approximations, let us start with a finite MDP with a set of nodes $\{s_n\}_{n=1}^M$ where M is the number of nodes. The state-value function $V(s)$ in this case is determined by a set of node values V_n for each node of the state grid. We can present this set using an “index-free” notation:

$$V(s) = \sum_{n=1}^M V_n \delta_{s,s_n}, \quad (9.52)$$

where δ_{s,s_n} is the Kronecker symbol:

$$\delta_{s,s_n} = \begin{cases} 1 & \text{if } s = s_n \\ 0 & \text{otherwise.} \end{cases} \quad (9.53)$$

Eq.(9.52) can be viewed as an approach to conveniently and simultaneously represent all values V_n of the value function on the grid in the form of a map $V(s)$ that points to a given node value V_n for any particular choice $s = s_n$.

Now we can write the same equation (9.52) in a more suggestive form as an expansion over a set of basis functions

$$V(s) = \sum_{n=1}^M V_n \delta_{s,s_n} = \sum_{n=1}^M V_n \phi_n(s), \quad (9.54)$$

with “one-hot” basis functions $\phi_n(s)$:

$$\phi_n(s) = \delta_{s,s_n} = \begin{cases} 1 & \text{if } s = s_n \\ 0 & \text{otherwise.} \end{cases} \quad (9.55)$$

The latter form helps to understand how this setting can now be generalized for a continuous state space. In a discrete-state representation (9.54), we use “one-hot” (Dirac-like) basis functions $\phi_n(s) = \delta_{s,s_n}$. We can now envision a transition to a continuous time limit as a process of adding new points to the grid, while at the same time keeping the size M of the sum in Eq. (9.54) by aggregating (taking partial sums) within some neighborhoods of a set of M node points. Each term in such a sum would be given by the product of an average mass of nodes V_n and a smoothed-out version of the original Dirac-like basis function of a finite MDP. Such partial aggregation of neighboring points produces a tractable approximation to the actual value function defined in a continuous limit. A function value at any point of a continuous space is now mapped onto a M -dimensional approximation. The quality of such finite-dimensional function approximation is determined by the number of terms in the expansion, as well as the functional form of the basis functions.

A smoothed-out localized basis could be constructed using, e.g., B-splines or Gaussian kernels, while for a multi-dimensional continuous-state case one could use multivariate B-splines or radial basis functions (RBFs). A set of one-dimensional B-spline basis functions is illustrated in Fig. 9.8. As one can see, B-splines produce well localized basis functions that differ from zero only on a limited segment of a total support region. Other alternatives, for example, polynomials or trigonometric functions, can also be considered for basis functions; however, they would be functions of a global, rather than local, variation.

Similar expansions can be considered for action-value functions $Q(s, a)$. Let us assume that we have a set of basis functions $\psi_k(s, a)$ with $k = 0, 1, \dots, K$ defined on the direct product $\mathcal{S} \times \mathcal{A}$. Respectively, we could approximate an action-value function as follows:

$$Q(s, a) = \sum_{k=0}^{K-1} \theta_k \psi_k(s, a). \quad (9.56)$$

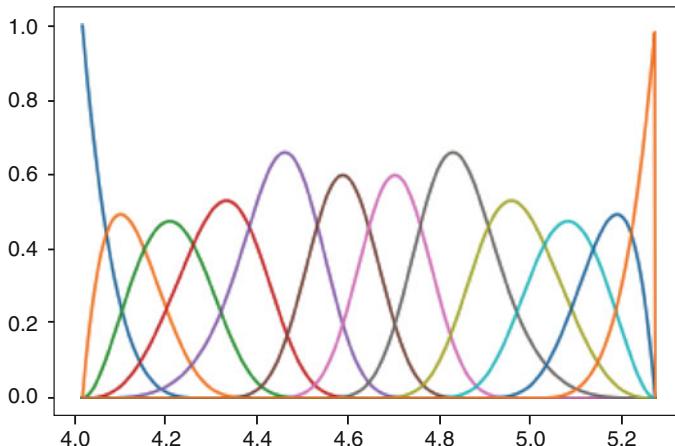


Fig. 9.8 B-spline basis functions

Here coefficients θ_k of such expansion can be viewed as free parameters. Respectively, we can find the values of these parameters which provide the best fit to the Bellman optimality equation. For a fixed and finite set of K basis functions $\psi_k(s, a)$, the problem of functional optimization of the action-value function $Q(s, a)$ is reduced by Eq. (9.56) to a much simpler problem of K -dimensional numeric optimization over parameters θ_k , irrespective of the actual dimensionality of the state space.

Note that having only a finite (and not too large) value of K , we can at best hope only for an approximate match between the optimal value function obtained in this way with a “true” optimal value function. The latter could in principle be obtained with the same basis function expansion (9.56), provided the set $\{\psi_k(s, a)\}$ is complete, by taking the limit $K \rightarrow \infty$.

Equation (9.56) thus provides an example of function approximation, where a function of interest is represented as expansion over a set of K basis functions $\psi_k(s, a)$, with coefficients θ_k being adjustable parameters. Such linear function representations are usually referred to as *linear architectures* in the machine learning literature. Functions of interest such as value functions and/or policy functions are represented and computed in these linear architecture methods as linear functions of parameters θ_k and basis functions $\psi_k(s, a)$.

The main advantage of linear architecture approaches is their relative robustness and computational simplicity. The possible amount of variation in functions being approximated is essentially determined by a possible amount of variation in basis functions, and thus can be explicitly controlled. Furthermore, because Eq. (9.56) is linear in θ , this can produce analytic solutions if a loss function is quadratic, or unique and easily computed numerical solutions when a loss function is non-quadratic but convex. Reinforcement learning methods with linear architectures are provably convergent.

On the other hand, the linear architecture approach is not free of drawbacks. The main one is that it gives no guidance on how to choose a good set of basis functions. For a bounded one-dimensional continuous state, it is not difficult to derive a good set of basis functions. For example, we could use a trigonometric basis, or splines, or even a polynomial basis. However, with multiple continuous dimensions, to find a good set of basis functions is non-trivial. This goes beyond reinforcement learning, and is generally known in machine learning as a feature construction (or extraction) problem.

One possible approach to deal with such cases is to use *non-linear architectures* that use generic function approximation tools such as trees or neural networks, in order to have flexible representations for functions of interest that do not rely on any pre-defined set of basis functions. In particular, deep reinforcement learning is obtained when one uses a deep neural network to approximate value functions or action policy (or both) in reinforcement learning tasks. We will discuss deep reinforcement learning below, after we present an off-line version of Q-learning that for both discrete and continuous state spaces while operating with mini-batches of data, instead of updating each observation.

5.7 Batch-Mode Q-Learning

Assume that we have a set of basis functions $\psi_k(s, a)$ with $k = 0, 1, \dots, K$ defined on the direct product $\mathcal{S} \times \mathcal{A}$, and the action-value function is represented by the linear expansion (9.56). Such a representation applies to both finite and continuous MDP problems—as we have seen above, a finite MDP case can be considered a special case of the linear architecture (9.56) with Dirac delta-functions taken as basis functions $\psi_k(s, a)$. Therefore, using the linear specification (9.56), we can provide a unified description of algorithms of Q-learning for both cases of finite and continuous MDPs.

Solving the Bellman optimality equation (9.20) now amounts to finding parameters θ_k . Clearly, if we want to find all $K > 1$ such parameters, observing just one data point in each iteration would be insufficient to determine them, or update their previous estimation in a unique and well-specified way. To this end, we need to tackle at least K observations (and, to avoid high variance estimations, a multiple of this number) to produce such estimate. In other words, we need to work in the setting of *batch-mode*, or off-line, reinforcement learning. With batch reinforcement learning, an agent does not have access to an environment, but rather can only operate with some historical data collected by observing actions of another agent over a period of time. Based on the law of large numbers, one can expect that whenever batch reinforcement learning can be used for training a reinforcement learning algorithm, it can provide estimators with lower variance than those obtained with a pure online learning.

To obtain a batch version of Q-learning, the one-step Bellman optimality equation (9.20) is interpreted as regression of the form

$$R_t(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}^*(s', a') = \sum_{k=0}^{K-1} \theta_k \psi_k(s, a) + \varepsilon_t, \quad (9.57)$$

where ε_t is a random noise at time t with mean zero. Parameters θ_k now appear as regression coefficients of the dependent variable $R_t(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}^*(s', a')$ on the regressors given by the basis functions $\psi_k(s, a)$. Equations (9.57) and (9.20) are equivalent in expectations, as taking the expectation of both sides of (9.57), we recover (9.20) with function approximation (9.56) used for the optimal Q-function $Q_t^*(s, a)$.

A batch data file consists of tuples $(s, a, r, s') = (s_t, a_t, r_t, s_{t+1})$ for $t = 0, \dots, T - 1$. Each tuple record (s, a, r, s') contains the current state $s = s_t$, action taken $a = a_t$, reward received $r = r_t$, and the next state s' . If we know the next-step action-value function $Q_{t+1}^*(s', a')$ as a function of state s' and action a' (via either an explicit formula or a numerical algorithm), the tuple record (s, a, r, s') could be used to view them as pairs (s, y) of a supervised regression with the independent variable $s = s_t$ and dependent variable $y := r + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}^*(s', a')$.

Note that there is a nuance here related to taking the max $\max_{a' \in \mathcal{A}}$ over all actions in the next time step. We will return to this point momentarily, but for now let us assume that this operation can be performed in one way or another, so that each tuple (s, a, r, s') can indeed be used as an observation for the regression (9.57).

Assume that for each time step t , we have samples from N trajectories.⁶ Using a conventional squared loss function, coefficients θ_k can be found by solving the standard least-square optimization problem:

$$\mathcal{L}_t(\theta) = \sum_{k=1}^N \left(R_t(s_k a_k, s'_k) + \gamma \max_{a' \in \mathcal{A}} Q_{t+1}^*(s', a') - \sum_{k=0}^{K-1} \theta_k \psi_k(s, a) \right)^2. \quad (9.58)$$

This is known as the Fitted Q Iteration (FQI) method. We will discuss an application of this method in the next chapter when we present reinforcement learning for option pricing.

Let us now address the challenge of computing the term $\max_{a' \in \mathcal{A}} Q_{t+1}^*(s', a')$ that appears in the regression (9.57) when we are only given samples of transitions in form of tuples (s, a, r, s') . One simple way would be to replace the theoretical maximum by an empirical maximum observed in the dataset. This would amount to using the same dataset to estimate both the optimal action and the optimal Q-function.

It turns out that such a procedure leads to an overestimation of $\max_{a'} Q_{t+1}^*(s, a')$ in the Bellman optimality equation (9.20), due to Jensen's inequality and convexity of the $\max(\cdot)$ function: $\mathbb{E}[\max f(x)] \leq \max \mathbb{E}[f(x)]$, where $f(x)$ is an arbitrary function. Indeed, by replacing expected maximum of $Q(s', a')$ by an empirical maximum, we replace the expected maximum $\mathbb{E}[\max_{a'} Q(s', a')]$ by

⁶These may be Monte Carlo trajectories or trajectories obtained from real-world data.

$\max_{a'} \mathbb{E}[Q(s', a')]$, and then further use a sample-based estimation of the inner expectation in the last expression. Due to Jensen's inequality, such replacement generally leads to overestimation of the action-value function. When repeated multiple times during iterations over time steps or during optimization over parameters, this overestimation can lead to distorted and sometimes even diverging value functions. This is known in the reinforcement learning literature as the *overestimation bias* problem.

There are two possible approaches to address a potential overestimation bias with Q-learning. One of them is to use two different datasets to train the action-value function and the optimal policy. But this is not directly implementable with Q-learning where policy is determined by the action-value function $Q(s, a)$. Instead of optimizing both the action-value function and the policy on different subsets of data, a method known as Double Q-learning (van Hasselt 2010) introduces two action-value functions $Q_A(s, a)$ and $Q_B(s, a)$. At each iteration, when presented with a new mini-batch of data, the Double Q-learning algorithm randomly chooses between an update of $Q_A(s, a)$ and update of $Q_B(s, a)$. If one chooses to update $Q_A(s, a)$, the optimal action is determined by finding a maximum a_\star of $Q_A(s', a')$, and then $Q_A(s, a)$ is updated using the TD error of $r + \gamma Q_B(s', a_\star) - Q_A(s, a)$. If, on the other hand, one chooses to update $Q_B(s, a)$, then the optimal action a_\star is calculated by maximizing $Q_B(s', a')$, and then updating $Q_B(s, a)$ using the TD error of $r + \gamma Q_A(s', a_\star) - Q_B(s, a)$. As was shown by van Hasselt (2010), action-value functions $Q_A(s, a)$ and $Q_B(s, a)$ converge in Double Q-learning to the same limit when the number of observation grows. The method avoids the overestimation bias problem of a naive sample-based Q-learning, though it can at times lead to *underestimation* of the action-value function. Double Q-learning is often used with model-free Q-learning using neural networks to represent an action-value function $Q_\theta(s, a)$ where θ is a set of parameters of the neural network.

Another possibility arises if the action-value function $Q(s, a)$ has a specific parametric form, so that the maximum over the next-step action a' can be performed analytically or numerically. In particular, for linear architectures (9.56) the maximum can be computed once the form of basis functions $\psi_k(s, a)$ and coefficients θ_k are fixed. With such an independent calculation of the maximum in the Bellman optimality equation, splitting a dataset into two separate datasets for learning the action-value function and optimal policy, as is done with Double Q-learning, can be avoided. As we will see in later chapters, such scenarios can be implemented for some problems in quantitative trading, including in particular option pricing.

► Bellman Equations and Non-expansion Operators

As we saw above, a non-analytic term in the Bellman optimality equation involving a max over all actions at the next step poses certain computational challenges. It turns out that this term can be relaxed using differentiable parameterized operators constructed in such a way that the “hard” max operator is recovered in a certain parametric limit. It turns out that operators

(continued)

of certain type, called *non-expansion* operators, can replace the *max* operator in a Bellman equation without loosing the existence of a solution, such as a fixed-point solution for a time-stationary MDP problem.

Let h be a real-valued function over a finite set I , and let \odot be a summary operator that maps values of function h onto a real number. The maximum operator $\max_{i \in I} h(i)$ and the minimum operators $\min_{i \in I} h(i)$ are examples of summary operators. A summary operator \odot is called a *non-expansion* if it satisfies two properties

$$\min_{i \in I} h(i) \leq \odot h(i) \leq \max_{i \in I} h(i) \quad (9.59)$$

and

$$|\odot h(i) - \odot h'(i)| \leq \max_{i \in I} |h(i) - h'(i)|, \quad (9.60)$$

where h' is another real-valued function over the same set. Some examples of non-expansion include the mean and max operator, as well as epsilon-greedy operator $\text{eps}_\varepsilon(\mathbf{X}) = \varepsilon \text{mean}(\mathbf{X}) + (1 - \varepsilon) \max(\mathbf{X})$.

As was shown by Littman and Szepasvari (1996), value iteration for the action-value function

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s, a, s') \max_{a' \in \mathcal{A}} \hat{Q}(s', a') \quad (9.61)$$

can be replaced by a generalized value iteration

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s, a, s') \odot_{a' \in \mathcal{A}} \hat{Q}(s', a') \quad (9.62)$$

which converges to a unique fixed point if operator \odot is a non-expansion with respect to the infinity norm:

$$|\odot \hat{Q}(s, a) - \odot \hat{Q}'(s, a)| \leq \max_a |\hat{Q}(s, a) - \hat{Q}'(s, a)| \quad (9.63)$$

for any \hat{Q}, \hat{Q}', s .

? Multiple Choice Question 3

Select all the following correct statements:

- a. Fitted Q Iteration is a method to accelerate *on-line* Q-learning.

- b. Similar to the DP approach, Fitted Q Iteration looks at only one trajectory at each update, so that Q-iteration fits better when extra noise from other trajectories is removed.
 - c. Fitted Q Iteration works only for discrete state-action spaces.
 - d. Fitted Q Iteration works only for continuous state-action spaces.
 - e. Fitted Q Iteration works for both discrete and continuous state-action spaces.
-

► Online Learning with MDPs

Recall that in Chap. 1, we presented the Multi-Armed Bandit (MAB) as an example of online sequential learning. Similar to the MAB formulation, for a more general setting of online learning with Markov Decision Processes,⁷ a training algorithm \mathcal{A} aims at minimization of the regret of \mathcal{A} defined as follows:

$$\mathbf{R}_T^{\mathcal{A}} = \mathcal{R}_T^{\mathcal{A}} - T\rho^*, \quad (9.64)$$

where $\mathcal{R}_T^{\mathcal{A}} = \sum_{t=0}^{T-1} R_{t+1}$ is the total reward received up to time T while following \mathcal{A} , and ρ^* stands for the optimal long-run average reward:

$$\rho^* = \max_{\pi} \rho^{\pi} = \max_{\pi} \sum_{s \in \mathcal{S}} \mu_{\pi}(s) R(s, \pi(s)), \quad (9.65)$$

where $\mu_{\pi}(s)$ is a stationary distribution of states induced by policy π . The problem of minimization of regret is clearly equivalent to maximization of the total reward. For a discussion of algorithms for online learning with the MDP, see Szepesvari (2010).

Online learning with MDP can be of interest in the financial context for certain tasks that may require real-time adjustments of policy following new data received, such as intraday trading. As we mentioned earlier, a combination of off-line and online learning using experience replay is often found to produce better and more stable behavior than pure online learning.

5.8 Least Squares Policy Iteration

Recall that for every MDP, there exists an optimal policy, π^* , which maximizes the expected, discounted return of every state. As discussed earlier in this chapter, policy iteration is a method of discovering this policy by iterating through a sequence of

⁷See Sect. 3 for further details of MDPs.

monotonically improving policies. Each iteration consists of two phases: (i) Value determination computes the state-action values for a policy, π by solving the above system; (ii) Policy improvement defines the next policy π' . These steps are repeated until convergence.

The Least Squares Policy Iteration (LSPI) (Lagoudakis and Parr 2003) is a model-free off-policy method that can efficiently use (and re-use at each iteration) sample experiences collected using any policy .

The LSPI method can be understood as a sample-based and model-free approximate policy iteration method that uses a linear architecture where an action-value function $Q_t(x_t, a_t)$ is sought as a linear expansion in a set of basis functions.

The LSPI approach proceeds as follows. Assume we have a set of K basis functions $\{\Psi_k(x, a)\}_{k=1}^K$. While particular choices for such basis functions will be presented below, in this section their specific form does not matter, as long as the set of basis function is expressive enough so that the true optimal action-value function approximately lies in a span of these basis functions. Provided such a set is fixed, we use a linear function approximation for the action-value function:

$$Q_t^\pi(x_t, a_t) = \mathbf{W}_t \Psi(x_t, a_t) = \sum_{k=1}^K W_{tk} \Psi_k(x_t, a_t). \quad (9.66)$$

Note that a dependence on a policy π enters this expression through a dependence of coefficients W_{ik} on π . The LSPI method can be thought of a process of finding an optimal policy via adjustments of weights W_{ik} . The policy π is a greedy policy that maximizes the action-value function:

$$a_t^\star(x_t) = \pi_t(x_t) = \operatorname{argmax}_a Q_t(x_t, a_t). \quad (9.67)$$

The LSPI algorithm continues iterating between computing coefficients W_{tk} (and hence the action-value function $Q_t(x_t, a_t)$) and computing the policy given the action-value function using Eq. (9.74). This is done for each time step, proceeding backward in time for $t = T - 1, \dots, 0$.

To find coefficients \mathbf{W}_t for a given time step, we first note that a linear Bellman equation (9.18) for a *fixed* policy π can be expressed in a form that only involve the action-value function, by noting that for an arbitrary policy π , we have

$$V_t^\pi(x_t) = Q_t^\pi(x_t, \pi(x_t)). \quad (9.68)$$

Using this in the Bellman equation (9.18), we write it in the following form:

$$Q_t^\pi(x_t, a_t) = R_t(x_t, a_t) + \gamma \mathbb{E}_t [Q_{t+1}^\pi(X_{t+1}, \pi(X_{t+1})) | x_t, a_t]. \quad (9.69)$$

Similar to Eq. (9.57), we can interpret Eq. (9.69) as regression of the form

$$R_t(x_t, a_t, x_{t+1}) + \gamma Q_{t+1}^\pi(x_{t+1}, \pi(x_{t+1})) = \mathbf{W}_t \Psi(x_t, a_t) + \varepsilon_t, \quad (9.70)$$

where $R_t(x_t, a_t, x_{t+1})$ is a random reward, and ε_t is a random noise at time t with mean zero. Assume we have access to sample transitions $(X_t^{(k)}, a_t^{(k)}, R_t^{(k)}, X_{t+1}^{(k)})$ with $k = 1, \dots, N$ for each $t = T-1, \dots, 0$. For a given policy π , coefficients \mathbf{W}_t can then be found by solving the following least squares optimization problem, which is similar to Eq. (9.58) above:

$$\begin{aligned} \mathcal{L}_t(\mathbf{W}_t) = & \sum_{k=1}^N \left(R_t\left(X_t^{(k)}, a_t^{(k)}, X_{t+1}^{(k)}\right) \right. \\ & \left. + \gamma Q_{t+1}^\pi\left(X_{t+1}^{(k)}, \pi\left(X_{t+1}^{(k)}\right)\right) - \mathbf{W}_t \Psi\left(X_t^{(k)}, a_t^{(k)}\right) \right)^2. \end{aligned} \quad (9.71)$$

For a solution of this equation, see Exercise 9.14.

Note that for an MDP with a finite state-action space, finding an optimal policy using (9.67) is straightforward, and achieved by enumeration of possible actions for each state. When the action space is continuous, it takes more effort. Consider, for example, the case where both the state and action spaces are one-dimensional continuous spaces. To use Eq. (9.67) in such setting, we discretize the range of values of x_t to a set of discrete values x_n . We can first compute the optimal action for these values, and then use splines to interpolate for the rest of values of x_t . For a given set of coefficients W_{tk} , the policy $\pi_t(x_t)$ is then represented by a spline-interpolated function.

Example 9.7 LSPI for Optimal Allocation

Recall from Chap. 1 the problem of an investor who starts with an initial wealth $W_0 = 1$ at time $t = 0$ and, at each period $t = 0, \dots, T-1$ allocates a fraction $u_t = u_t(S_t)$ of the total portfolio value to the risky asset, and the remaining fraction $1 - u_t$ is invested in a risk-free bank account that pays a risk-free interest rate $r_f = 0$. If the wealth process is self-financing and the one-step rewards R_t for $t = 0, \dots, T-1$ are the risk-adjusted portfolio returns

$$R_t = r_t - \lambda \text{Var}[r_t | S_t] = u_t \phi_t - \lambda u_t^2 \text{Var}[\phi_t | S_t] \quad (9.72)$$

then the optimal investment problem for $T-1$ steps is given by

$$V^\pi(s) = \max_{u_t} \mathbb{E} \left[\sum_{t=0}^T R_t \middle| S_t = s \right] = \max_{u_t} \mathbb{E} \left[\sum_{t=0}^T u_t \phi_t - \lambda u_t^2 \text{Var}[\phi_t | S_t] \middle| S_t = s \right]. \quad (9.73)$$

(continued)

Example 9.7 (continued)

where we allow for short selling in the ETF (i.e., $u_t < 0$) and borrowing of cash $u_t > 1$. We apply the LSPI algorithm to $N = 2000$ simulated stock prices over $T = 10$ periods: $\{\{S_t^{(i)}\}_{i=1}^N\}_{t=1}^T$. At each time period, we construct a basis $\{\Psi_k(s, a)\}_{k=1}^K$ over the state-action space using $K = 256$ B-spline basis functions, where $s \in [\min(\{S_t^{(i)}\}_{i=1}^N), \max(\{S_t^{(i)}\}_{i=1}^N)]$ and $a \in [-1, 1]$.

Note that in this particular simple problem, the actions are independent of the state space and hence the basis construction over state-action space is not really needed. However, our motive here is to show that we can obtain an estimate close to the exact solution, $u_t^* = \frac{\mathbb{E}[\phi_t]}{2\lambda \text{Var}[\phi_t]}$ using a more general methodology.

a_{T-1} is initialized with uniform random samples at time step $t = T - 1$. In subsequent time steps, a_t , $t \in \{T - 1, T - 2, \dots, 0\}$, we initialize with the previous optimal action, $a_t = a_{t+1}^*$. LSPI updates the policy iteratively $\pi^{k-1} \rightarrow \pi^k$ until convergence of the value function. The Q-function is maximized over a gridded state-action space, Ω^h , with 200 stock values and 20 action values, to give the optimal action

$$a_t^k(s) = \pi_t^k(s) = \underset{a}{\operatorname{argmax}} \quad Q_t^{\pi^{k-1}}(s, a), \quad (s, a) \in \Omega^h. \quad (9.74)$$

For each time step, LSPI updates the policy π^k until the following stopping criterion is satisfied $\|V^{\pi^k} - V^{\pi^{k-1}}\|_2 \leq \tau$ where $\tau = 1 \times 10^{-6}$. The optimal allocation using the LSPI algorithm (red) is compared against the exact solution (blue) in Fig. 9.9. The implementation of the LSPI and its application to this optimal allocation problem is given in the Python notebook `ML_in_Finance_LSPI_Markowitz.ipynb`.

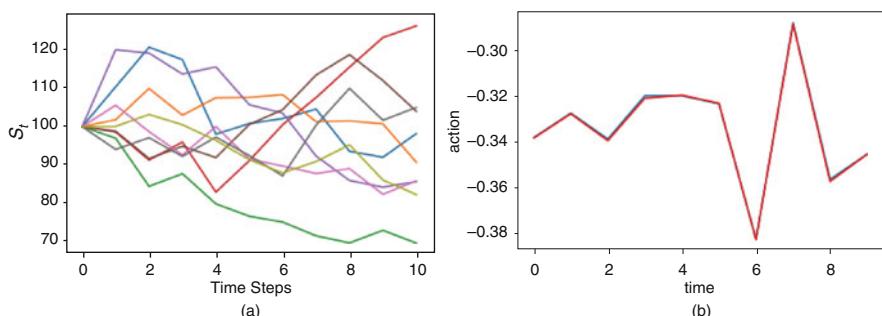


Fig. 9.9 Stock prices are simulated using an Euler scheme over a one year horizon. At each of ten periods, the optimal allocation is estimated using the LSPI algorithm (red) and compared against the exact solution (blue). **(a)** Stock price simulation. **(b)** Optimal allocation

5.9 Deep Reinforcement Learning

A good choice of basis functions that could be used in linear architectures (9.56) may be a hard problem for many practical applications, especially if the dimensionality of data grows, or if data become highly non-linear, or both. This problem is also known as the feature engineering problem, and it is common for all types of machine learning, rather than being specific to reinforcement learning. Learning representative features is an interesting and actively researched topic in the machine learning literature, and various supervised and unsupervised algorithms have been suggested to address such tasks.

Instead of pursuing hand-engineered or algorithm-driven features defined in general as parameter-based functional transforms of original data, we can resort to universal function approximation methods such as trees or neural networks considered as parameterized “black-box” algorithms. In particular, deep reinforcement learning approaches rely on multi-level neural networks to represent value functions and/or policy functions. For example, if an action-value function $Q(s, a)$ is represented by a multilayer neural network, one way of thinking about it would be in terms of the linear architecture specification (9.56) where parameters θ_k represent the weight of a last linear layer of a neural network, while the previous layer generates certain “black-box”-type basis functions $\psi_k(s, a)$ that can be parameterized in terms of their own parameters θ' . This approach may be advantageous when an action-value function is highly non-linear and no clear-cut choice of a good set of basis function can be immediately suggested. In particular, functions of high variations appear in analysis of images, videos, and video games. For such applications, using deep neural networks as function approximation is very useful. A strong push to this area of research was initiated by Google’s DeepMind’s work on using deep Q-learning for playing Atari video games.

5.9.1 Preliminaries

Since we cannot reasonably learn and store a Q value for each state-action tuple when the state space is continuous, we will represent our Q values as a function $\hat{q}(s, a, \mathbf{w})$ where \mathbf{w} are parameters of the function (typically, a neural network’s weights and bias). In this *approximation* setting, our update rule becomes

$$\mathbf{w} = \mathbf{w} + \alpha \left(r + \gamma \max_{a' \in A} \hat{q}(s', a', \mathbf{w}) - \hat{q}(s, a, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(s, a, \mathbf{w}). \quad (9.75)$$

In other words, we seek to minimize

$$L(\mathbf{w}) = \mathbf{E}_{s, a, r, s'} \left[r + \gamma \max_{a' \in A} \hat{q}(s', a', \mathbf{w}) - \hat{q}(s, a, \mathbf{w}) \right]^2. \quad (9.76)$$

5.9.2 Target Network

DeepMind (Mnih et al. 2015) maintain two sets of parameters, \mathbf{w} (to compute $\hat{q}(s, a)$) and \mathbf{w}^- (target network, to compute $\hat{q}(s', a')$) s.t. our update rule becomes

$$\mathbf{w} = \mathbf{w} + \alpha \left(r + \gamma \max_{a' \in A} \hat{q}(s', a', \mathbf{w}^-) - \hat{q}(s, a, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(s, a, \mathbf{w}). \quad (9.77)$$

The target network's parameters are updated with the Q-network's parameters occasionally and are kept fixed between individual updates. Note that when computing the update, we do not compute gradients with respect to \mathbf{w}^- (these are considered fixed weights).

5.9.3 Replay Memory

As we play, we store our transitions (s, a, r, s') in a buffer. Old examples are deleted as we store new transitions. To update our parameters, we *sample* a mini-batch from the buffer and perform a stochastic gradient descent update.

ϵ -Greedy Exploration Strategy

During training, we use an ϵ -greedy heuristic strategy. DeepMind (Mnih et al. 2015) decrease ϵ from 1 to 0.1 during the first million steps. At test time, the agent chooses a random action with probability $\epsilon = 0.05$.

$$\pi(a|s) = \begin{cases} 1 - \epsilon, & a = \operatorname{argmax}_a Q_t(s, a) \\ \epsilon/|A|, & a \neq \operatorname{argmax}_a Q_t(s, a) \end{cases}$$

There are several points to note:

- \mathbf{w} updates every `learning_freq` steps by using a mini-batch of experiences sampled from the replay buffer.
- DeepMind's deep Q-network takes as input the state s and outputs a vector of size = number of actions. In our environment, we have $|A|$ actions, thus $\hat{q}(s, \mathbf{w}) \in \mathbb{R}^{|A|}$.
- The input of the deep Q-network can be based on both the current and history of observations of the environment.

The practice of using Deep Q-learning on finance problems is problematic. The sheer number of parameters that need to be tuned and configured renders the approach much more complex than Q-learning, LSPI, and of course deep learning in a supervised learning setting. However, deep Q-learning is one of the few approaches which scales to high-dimensional discrete and/or continuous state and action spaces.

6 Summary

This chapter has introduced the reader to reinforcement learning, with some toy examples of how it is useful for solving problems in finance. The emphasis of the chapter is on understanding the various algorithms and RL approaches. The reader should check the following learning objectives:

- Gain familiarity with Markov Decision Processes;
- Understand the Bellman equation and classical methods of dynamic programming;
- Gain familiarity with the ideas of reinforcement learning and other approximate methods of solving MDPs;
- Understand the difference between off-policy and on-policy learning algorithms; and
- Gain insight into how RL is applied to optimization problems in asset management and trading.

The next chapter will present much more in depth examples of how RL is applied in more realistic financial models.

7 Exercises

Exercise 9.1

Consider an MDP with a reward function $r_t = r(s_t, a_t)$. Let $Q^\pi(s, a)$ be an action-value function for policy π for this MDP, and $\pi_*(a|s) = \arg \max_\pi Q^\pi(s, a)$ be an optimal greedy policy. Assume we define a new reward function as an affine transformation of the previous reward: $\tilde{r}(t) = wr_t + b$ with constant parameters b and $w > 0$. How does the new optimal policy $\tilde{\pi}_*$ relate to the old optimal policy π_* ?

Exercise 9.2

With True/False questions, give a short explanation to support your answer.

- True/False: Value iteration always find the optimal policy, when run to convergence. [3]
- True/False: Q-learning is an on-policy learning (value iteration) algorithm and estimates updates to the action-value function, $Q(s, a)$ using actions taken under the current policy π . [5]
- For Q-learning to converge we need to correctly manage the exploration vs. exploitation tradeoff. What property needs to hold for the exploration strategy? [4]
- True/False: Q-learning with linear function approximation will always converge to the optimal policy. [2]

Table 9.4 The reward function depends on fund wealth w and time

w	t_0	t_1
2	0	0
1	0	-10

Exercise 9.3*

Consider the following toy **cash buffer problem**. An investor owns a stock, initially valued at $S_{t_0} = 1$, and must ensure that their wealth (stock + cash) is not less than a certain threshold K at time $t = t_1$. Let $W_t = S_t + C_t$ denote their at time t , where C_t is the total cash in the portfolio. If the wealth $W_{t_1} < K = 2$ then the investor is penalized with a -10 reward.

The investor chooses to inject either 0 or 1 amounts of cash with a respective penalty of 0 or -1 (which is not deducted from the fund).

Dynamics The stock price follows a discrete Markov chain with $P(S_{t+1} = s | S_t = s) = 0.5$, i.e. with probability 0.5 the stock remains the same price over the time interval. $P(S_{t+1} = s + 1 | S_t = s) = P(S_{t+1} = s - 1 | S_t = s) = 0.25$. If the wealth moves off the grid it simply bounces to the nearest value in the grid at that time. The states are grid squares, identified by their row and column number (row first). The investor always starts in state (1,0) (i.e., the initial wealth $W_{t_0} = 1$ at time $t_0 = 0$ —there is no cash in the fund) and both states in the last column (i.e., at time $t = t_1 = 1$) are terminal (Table 9.4).

Using the Bellman equation (with generic state notation), give the first round of value iteration updates for each state by completing the table below. You may ignore the time value of money, i.e. set $\gamma = 1$.

$$V_{i+1}(s) = \max_a \left(\sum_{s'} T(s, a, s') (R(s, a, s') + \gamma V_i(s')) \right)$$

(w,t)	(1,0)	(2,0)
$V_0(w)$	0	0
$V_1(w)$?	NA

Exercise 9.4*

Consider the following toy **cash buffer problem**. An investor owns a stock, initially valued at $S_{t_0} = 1$, and must ensure that their wealth (stock + cash) does not fall below a threshold $K = 1$ at time $t = t_1$. The investor can choose to either sell the stock or inject more cash, but not both. In the former case, the sale of the stock at time t results in an immediate cash update s_t (you may ignore transactions costs). If the investor chooses to inject a cash amount $c_t \in \{0, 1\}$, there is a corresponding penalty of $-c_t$ (which is not taken from the fund).

Let $W_t = S_t + C_t$ denote their wealth at time t , where C_t is the total cash in the portfolio.

Dynamics The stock price follows a discrete Markov chain with $P(S_{t+1} = s | S_t = s) = 0.5$, i.e. with probability 0.5 the stock remains the same price over the time interval. $P(S_{t+1} = s + 1 | S_t = s) = P(S_{t+1} = s - 1 | S_t = s) = 0.25$. If the wealth moves off the grid it simply bounces to the nearest value in the grid at that time. The states are grid squares, identified by their row and column number (row first). The investor always starts in state $(1,0)$ (i.e., the initial wealth $W_{t_0} = 1$ at time $t_0 = 0$ —there is no cash in the fund) and both states in the last column (i.e., at time $t = t_1 = 1$) are terminal.

Using the Bellman equation (with generic state notation), give the first round of value iteration updates for each state by completing the table below. You may ignore the time value of money, i.e. set $\gamma = 1$.

$$V_{i+1}(s) = \max_a \left(\sum_{s'} T(s, a, s') (R(s, a, s') + \gamma V_i(s')) \right)$$

(w,t)	(0,0)	(1,0)
$V_0(w)$	0	0
$V_1(w)$?	?

Exercise 9.5*

Deterministic policies such as the greedy policy $p\pi_\star(a|s) = \arg \max_\pi Q^\pi(s, a)$ are invariant with respect to a shift of the action-value function by an arbitrary function of a state $f(s)$: $\pi_\star(a|s) = \arg \max_\pi Q^\pi(s, a) = \arg \max_\pi \tilde{Q}^\pi(s, a)$ where $\tilde{Q}^\pi(s, a) = Q^\pi(s, a) - f(s)$. Show that this implies that the optimal policy is also invariant with respect to the following transformation of an original reward function $r(s_t, a_t, s_{t+1})$:

$$\tilde{r}(s_t, a_t, s_{t+1}) = r(s_t, a_t, s_{t+1}) + \gamma f(s_{t+1}) - f(s_t).$$

This transformation of a reward function is known as *reward shaping* (Ng, Russell 1999). It has been used in reinforcement learning to accelerate learning in certain settings. In the context of inverse reinforcement learning, reward shaping invariance has far-reaching implications, as we will discuss later in the book.

Exercise 9.6**

Define the occupancy measure $\rho_\pi: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ by the relation

Table 9.5 The reward function depends on fund wealth w and time

w	t_0	t_1
1	0	0
0	0	-10

$$\rho_\pi(s, a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s|\pi),$$

where $\Pr(s_t = s|\pi)$ is the probability density of the state $s = s_t$ at time t following policy π . The occupancy measure $\rho_\pi(s, a)$ can be interpreted as an unnormalized density of state-action pairs. It can be used, e.g., to specify the value function as an expectation value of the reward: $V = \langle r(s, a) \rangle_\rho$.

- a. Compute the policy in terms of the occupancy measure ρ_π .
- b. Compute a normalized occupancy measure $\tilde{\rho}_\pi(s, a)$. How different the policy will be if we used the normalized measure $\tilde{\rho}_\pi(s, a)$ instead of the unnormalized measure ρ_π ?

Exercise 9.7**

Theoretical models for reinforcement learning typically assume that rewards $r_t := r(s_t, a_t, s_{t+1})$ are bounded: $r_{min} \leq r_t \leq r_{max}$ with some fixed values r_{min}, r_{max} . On the other hand, some models of rewards used by practitioners may produce (numerically) unbounded rewards. For example, with linear architectures, a popular choice of a reward function is a linear expansion $r_t = \sum_{k=1}^K \theta_k \Psi_k(s_t, a_t)$ over a set of K basis functions Ψ_k . Even if one chooses a set of bounded basis functions, this expression may become unbounded via a choice of coefficients θ_t .

- a. Use the policy invariance under linear transforms of rewards (see Exercise 9.1) to equivalently formulate the same problem with rewards that are bounded to the unit interval $[0, 1]$, so they can be interpreted as probabilities.
- b. How could you modify a linear unbounded specification of reward $r_\theta(s, a, s') = \sum_{k=1}^K \theta_k \Psi_k(s, a, s')$ to a bounded reward function with values in a unit interval $[0, 1]$?

Exercise 9.8

Consider an MDP with a finite number of states and actions in a real-time setting where the agent learns to act optimally using the ε -greedy policy. The ε -greedy policy amounts to taking an action $a_* = \operatorname{argmax}_{a'} Q(s, a')$ in each state s with probability $1 - \varepsilon$, and taking a random action with probability ε . Will SARSA and Q-learning converge to the same solution under such policy, using a constant value of ε ? What will be different in the answer if ε decays with the epoch, e.g. as $\varepsilon_t \sim 1/t$?

Exercise 9.9

Consider the following single-step random cost (negative reward)

$$C(s_t, a_t, s_{t+1}) = \eta a_t + (K - s_{t+1} - a_t)_+,$$

where η and K are some parameters. You can use such a cost function to develop an MDP model for an agent learning to invest. For example, s_t can be the current assets in a portfolio of equities at time t , a_t be an additional cash added to or subtracted from the portfolio at time t , and s_{t+1} be the portfolio value at the end of time interval

$[t, t + 1]$. The second term is an option-like cost of a total portfolio (equity and cash) shortfall by time $t + 1$ from a target value K . Parameter η controls the relative importance of paying costs now as opposed to delaying payment.

- What is the corresponding expected cost for this problem, if the expectation is taken w.r.t. to the stock prices and a_t is treated as deterministic?
- Is the expected cost a convex or concave function of the action a_t ?
- Can you find an optimal one-step action a_t^* that minimizes the one-step expected cost?

Hint: For Part (i), you can use the following property:

$$\frac{d}{dx} [y - x]_+ = \frac{d}{dx} [(y - x)H(y - x)],$$

where $H(x)$ is the Heaviside function.

Exercise 9.10

Exercise 9.9 presented a simple single-period cost function that can be used in the setting of model-free reinforcement learning. We can now formulate a model based formulation for such an option-like reward. To this end, we use the following specification of the random end-of-period portfolio state:

$$\begin{aligned}s_{t+1} &= (1 + r_t)s_t \\ r_t &= G(\mathbf{F}_t) + \varepsilon_t.\end{aligned}$$

In words, the initial portfolio value $s_t + a_t$ in the beginning of the interval $[t, t + 1]$ grows with a random return r_t given by a function $G(\mathbf{F}_t)$ of factors \mathbf{F}_t corrupted by noise ε with $\mathbb{E}[\varepsilon] = 0$ and $\mathbb{E}[\varepsilon^2] = \sigma^2$.

- Obtain the form of expected cost for this specification in Exercise 9.9.
- Obtain the optimal single-step action for this case.
- Compute the sensitivity of the optimal action with respect to the i -th factor \mathbf{F}_{it} assuming the sigmoid link function $G(\mathbf{F}_t) = \sigma(\sum_i \omega_i F_{it})$ and a Gaussian noise ε_t .

Exercise 9.11

Assuming a discrete set of actions $a_t \in \mathcal{A}$ of dimension K show that deterministic policy optimization by greedy policy of Q-learning $Q(s_t, a_t^*) = \max_{a_t \in \mathcal{A}} Q(s_t, a_t)$ can be equivalently expressed as maximization over a set *probability distributions* $\pi(a_t)$ with probabilities π_k for $a_t = A_k$, $k = 1, \dots, K$ (this relation is known as Fenchel duality):

$$\max_{a_t \in \mathcal{A}} Q(s_t, a_t) = \max_{\{\pi\}_k} \sum_{k=1}^K \pi_k Q(s_t, A_k) \quad \text{s.t. } 0 \leq \pi_i \leq 1, \sum_{k=1}^K \pi_k = 1.$$

Exercise 9.12**

The reformulation of a deterministic policy search in terms of search over probability distributions given in Exercise 9.11 is a mathematical identity where the end result is still a deterministic policy. We can convert it to a probabilistic policy search if we modify the objective function

$$\max_{a_t \in \mathcal{A}} Q(s_t, a_t) = \max_{\{\pi\}_k} \sum_{k=1}^K \pi_k Q(s_t, A_k) \quad \text{s.t. } 0 \leq \pi_i \leq 1, \sum_{k=1}^K \pi_k = 1$$

by adding to it a KL divergence of the policy π with some reference (“prior”) policy ω :

$$G^\star(s_t, a_t) = \max_{\pi} \sum_{k=1}^K \pi_k Q(s_t, A_k) - \frac{1}{\beta} \sum_{k=1}^K \pi_k \log \frac{\pi_k}{\omega_k},$$

where β is a regularization parameter controlling the relative importance of the two terms that enforce, respectively, maximization of the action-value function and a preference for a previous reference policy ω with probabilities ω_k . When parameter $\beta < \infty$ is finite, this produces a stochastic rather than deterministic optimal policy $\pi^\star(a|s)$.

Find the optimal policy $\pi^\star(a|s)$ from the entropy-regularized functional $G(s_t, a_t)$ (Hint: use the method of Lagrange multipliers to enforce the normalization constraint $\sum_k \pi_k = 1$).

Exercise 9.13**

Regularization by KL-divergence with a reference distribution ω introduced in the previous exercise can be extended to a multi-period setting. This produces maximum entropy reinforcement learning which augments the standard RL reward by an additional entropy penalty term in the form of KL divergence. The optimal value function in MaxEnt RL is

$$F^\star(s) = \max_{\pi} \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t, s_{t+1}) - \frac{1}{\beta} \log \frac{\pi(a_t|s_t)}{\pi_0(a_t|s_t)} \right) \middle| s_0 = s \right], \quad (9.78)$$

where $\mathbb{E}[\cdot]$ stands for an average under a stationary distribution $\rho_\pi(a) = \sum_s \mu_\pi(s) \pi(a|s)$ where $\mu_\pi(s)$ is a stationary distribution over states induced by the policy π , and π_0 is some reference policy. Show that the optimal policy for this entropy-regularized MDP problem has the following form:

$$\pi^\star(a|s) = \frac{1}{Z_t} \pi_0(a_t|s_t) e^{\beta G_t^\pi(s_t, a_t)}, \quad Z_t \equiv \sum_{a_t} \pi_0(a_t|s_t) e^{\beta G_t^\pi(s_t, a_t)}, \quad (9.79)$$

where $G_t^\pi(s_t, a_t) = \mathbb{E}^\pi [r(s_t, a_t, s_{t+1})] + \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) F_{t+1}^\pi(s_{t+1})$. Check that the limit $\beta \rightarrow \infty$ reproduces the standard deterministic policy, that is $\lim_{\beta \rightarrow \infty} V^\star(s) = \max_\pi V^\pi(s)$, while in the opposite limit $\beta \rightarrow 0$ we obtain a random and uniform policy. We will return to entropy-regularized value-based RL and stochastic policies such as (9.79) (which are sometimes referred to as Boltzmann policies) in later chapters of this book.

Exercise 9.14*

Show that the solution for the coefficients W_{tk} in the LSPI method (see Eq. (9.71)) is

$$\mathbf{W}_t^\star = \mathbf{S}_t^{-1} \mathbf{M}_t,$$

where \mathbf{S}_t is a matrix and \mathbf{M}_t is a vector with the following elements:

$$S_{nm}^{(t)} = \sum_{k=1}^N \Psi_n \left(X_t^{(k)}, a_t^{(k)} \right) \Psi_m \left(X_t^{(k)}, a_t^{(k)} \right)$$

$$M_n^{(t)} = \sum_{k=1}^N \Psi_n \left(X_t^{(k)}, a_t^{(k)} \right) \left(R_t \left(X_t^{(k)}, a_t^{(k)}, X_{t+1}^{(k)} \right) + \gamma Q_{t+1}^\pi \left(X_{t+1}^{(k)}, \pi \left(X_{t+1}^{(k)} \right) \right) \right).$$

Exercise 9.15**

Consider the Boltzmann weighted average of a function $h(i)$ defined on a binary set $I = \{1, 2\}$:

$$\text{Boltz}_\beta h(i) = \sum_{i \in I} h(i) \frac{e^{\beta h(i)}}{\sum_{i \in I} e^{\beta h(i)}}$$

- a. Verify that this operator smoothly interpolates between the max and the mean of $h(i)$ which are obtained in the limits $\beta \rightarrow \infty$ and $\beta \rightarrow 0$, respectively.
- b. By taking $\beta = 1$, $h(1) = 100$, $h(2) = 1$, $h'(1) = 1$, $h'(2) = 0$, show that Boltz_β is not a non-expansion.
- c. (Programming) Using operators that are not non-expansions can lead to a loss of a solution in a generalized Bellman equation. To illustrate such phenomenon, we use the following simple example. Consider the MDP problem on the set $I = \{1, 2\}$ with two actions a and b and the following specification: $p(1|1, a) = 0.66$, $p(2|1, a) = 0.34$, $r(1, a) = 0.122$ and $p(1|1, b) = 0.99$, $p(1|1, b) = 0.01$, $r(1, b) = 0.033$. The second state is absorbing with $p(1|2) = 0$, $p(2|2) = 1$. The discount factor is $\gamma = 0.98$. Assume we use the Boltzmann policy

$$\pi(a|s) = \frac{e^{\beta \hat{Q}(s, a)}}{\sum_a e^{\beta \hat{Q}(s, a)}}.$$

Show that the SARSA algorithm

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \left[r(s, a) + \gamma \hat{Q}(s', a') - \hat{Q}(s, a) \right],$$

where a , a' are drawn from the Boltzmann policy with $\beta = 16.55$ and $\alpha = 0.1$, leads to oscillating solutions for $\hat{Q}(s_1, a)$ and $\hat{Q}(s_1, a')$ that do not achieve stable states with an increased number of iterations.

Exercise 9.16**

An alternative continuous approximation to the intractable *max* operator in the Bellman optimality equation is given by the *mellowmax* function (Asadi and Littman 2016)

$$mm_\omega(\mathbf{X}) = \frac{1}{\omega} \log \left(\frac{1}{n} \sum_{i=1}^n e^{\omega x_i} \right)$$

- a. Show that the mellowmax function recovers the *max* function in the limit $\omega \rightarrow \infty$.
- b. Show that mellowmax is a non-expansion.

Appendix

Answers to Multiple Choice Questions

Question 1

Answer: 2, 4.

Question 2

Answer: 2, 4

Question 3

Answer: 5

Python Notebooks

The notebooks provided in the accompanying source code repository accompany many of the examples in this chapter, including Q-learning and SARSA for the financial cliff walking problem, the market impact problem, and electronic market making. The repository also includes an example implementation of the LSPI algorithm for optimal allocation in a Markowitz portfolio. Further details of the notebooks are included in the README.md file.

References

- Asadi, K., & Littman, M. L. (2016). An alternative softmax operator for reinforcement learning. *Proceedings of ICML*.
- Bellman, R. E. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.
- Bertsekas, D. (2012). *Dynamic programming and optimal control* (vol. I and II), 4th edn. Athena Scientific.
- Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, 4, 1107–1149.
- Littman, M. L., & Szepesvari, S. (1996). A generalized reinforcement-learning model: convergence and applications. In *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96)*, Bari, Italy.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.
- Robbins, H., & Monro, S. (1951). A stochastic approximation method. *Ann. Math. Statistics*, 22, 400–407.
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*, 2nd edn. MIT.
- Szepesvari, S. (2010). *Algorithms for reinforcement learning*. Morgan & Claypool.
- Thompson, W. R. (1935). On a criterion for the rejection of observations and the distribution of the ratio of deviation to sample standard deviation. *Ann. Math. Statist.*, 6(4), 214–219.
- Thompson, W. R. (1993). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3), 285–94.
- van Hasselt, H. (2010). Double Q-learning. *Advances in Neural Information Processing Systems*. <http://papers.nips.cc/paper/3964-double-q-learning.pdf>.

Chapter 10

Applications of Reinforcement Learning



This chapter considers real-world applications of reinforcement learning in finance, as well as further advances in the theory presented in the previous chapter. We start with one of the most common problems of quantitative finance, which is the problem of optimal portfolio trading in discrete time. Many practical problems of trading or risk management amount to different forms of dynamic portfolio optimization, with different optimization criteria, portfolio composition, and constraints. This chapter introduces a reinforcement learning approach to option pricing that generalizes the classical Black–Scholes model to a data-driven approach using Q-learning. It then presents a probabilistic extension of Q-learning called G-learning and shows how it can be used for dynamic portfolio optimization. For certain specifications of reward functions, G-learning is semi-analytically tractable and amounts to a probabilistic version of linear quadratic regulators (LQR). Detailed analyses of such cases are presented, and show their solutions with examples from problems of dynamic portfolio optimization and wealth management.

1 Introduction

In this chapter, we consider real-world applications of reinforcement learning in finance. We start with one of the most common problems of quantitative finance, which is the problem of optimal portfolio trading. Many practical problems of trading or risk management amount to different forms of dynamic portfolio optimization, with different optimization criteria, portfolio composition, and constraints. For example, the problem of optimal stock execution can be viewed as a problem of optimal dynamic management of a portfolio of stocks of the same company, with the objective being minimization of slippage costs of selling the stock. A more traditional example of dynamic portfolio optimization is given by asset managers and mutual or pension funds who usually manage investment

portfolios over long time horizons (months or years). Intra-day trading which is more typical of hedge funds can also be thought of as a dynamic portfolio optimization problem with a different portfolio choice, time step, constraints, and so on. In addition to different time horizons and objective functions, details of a portfolio optimization problem determine choices for features and hence for a state description. For example, management of long-horizon investment portfolios typically involves macroeconomic factors but not the limit order book data, while for intra-day trading it is the opposite case.

Dynamic portfolio management is a problem of stochastic optimal control where control variables are represented by changes in positions in different assets in a portfolio made by a portfolio manager, and state variables describe the current composition of the portfolio, prices of its assets, and possibly other relevant features including market indices, bid–ask spreads, etc. If we consider a large market player whose trade can move the market, actions of such trader may produce a feedback loop effect. The latter is referred to in the financial literature as a “market impact effect.”

All the above elements of dynamic portfolio management make it suitable for applying methods of dynamic programming and reinforcement learning. While the previous chapter introduced the main concepts and methods of reinforcement learning, here we want to take a more detailed look at practical applications for portfolio management problems.

When viewed as problems of optimal control to be addressed using reinforcement learning, such problems typically have a very high-dimensional state-action space. Indeed, even if we constrain ourselves by actively traded US stocks, we get around three thousands stocks. If we add to this other assets such as futures, exchange-traded funds, government and corporate bonds, etc., we may end up with state spaces of dimensions of many thousands. Even in a more specialized case of an equity fund whose objective is to beat a given benchmark portfolio, the investment universe may be tens or hundreds of stocks. This means that such applications of reinforcement learning in finance have to handle (very) high-dimensional and typically continuous (or approximately continuous) state-action spaces.

Clearly, such high-dimensional RL problems are far more complex than simple low-dimensional examples typically used to test reinforcement learning methods, such as inverted pendulum or cliff walking problems described in the Sutton–Barto book, or the “financial cliff walking” problem presented in the previous chapter. Modern RL methods applied to problems outside of finance typically operate with action space dimensions measured in tens but not hundreds or thousands, and typically have a far larger signal-to-noise ratio than financial problems. Low signal-to-noise ratios and potentially very high dimensionality are therefore two marked differences of applications of reinforcement learning in finance as opposed to applications to video games and robotics.

As high-dimensional optimal control problems are harder than low-dimensional ones, we first want to explore applications of reinforcement learning with low-dimensional portfolio optimization problems. Such an approach both sets the

grounds for more complex high-dimensional applications and can be of independent interest when applied to practically interesting problems falling in this general class of dynamic portfolio optimization.

The first problem that we address in this chapter is exactly of this kind: it is both low-dimensional and of practical interest on its own, rather than being a toy example for a multi-asset portfolio optimization. Namely, we will consider the classical problem of option pricing, in a formulation that closely resembles the framework of the celebrated Black–Scholes–Merton (BSM) model , also known as the Black–Scholes (BS) model, one of the cornerstones of modern quantitative finance (Black and Scholes 1973; Merton 1974).

Chapter Objectives

This chapter will present a few practical cases of using reinforcement learning in finance:

- RL for option pricing and optimal hedging (QLBS);
- G-learning for stock portfolios and linear quadratic regulators;
- RL for optimal consumption using G-learning; and
- RL for portfolio optimization using G-learning.

The chapter is accompanied by two notebooks implementing the QLBS model for option pricing and hedging, and G-learning for wealth management. See Appendix “Python Notebooks” for further details.

2 The QLBS Model for Option Pricing

The BSM model was initially developed for the so-called plain vanilla European call and put options. A European call option is a contract that allows a buyer to obtain a given stock at some future time T for a fixed price K . If S_T is the stock price at time T , then the payoff to the option buyer at time T is $(S_T - K)_+$. Similarly, a buyer of a European put option has a right to sell the stock at time T for a fixed price K , with a terminal payoff of $(K - S_T)_+$. European call and put options are among the simplest and most popular types of *financial derivatives* whose value is *derived* from (or driven by) the underlying stock (or more generally, the underlying asset).

The core idea of the BSM model is that options can be priced using the *relative value* approach to asset pricing, which prices assets in terms of other tradable assets. The relative pricing method for options is known as *dynamic option replication*. It is based on the observation that an option payoff depends only on the price of a stock at expiry of the option. Therefore, if we neglect other sources of uncertainty such as stochastic volatility, the option value at arbitrary times before the expiry should only depend on the stock value. This makes it possible to mimic the option using a simple portfolio made of the underlying stock and cash, which is called the hedge

portfolio. The hedge portfolio is dynamically managed by continuously rebalancing its wealth between the stock and cash. Moreover, this is done in a self-financing way, meaning that there are no cash inflows/outflows in the portfolio except at the time of inception. The objective of dynamic replication is to mimic the option using the hedge portfolio as closely as possible.

In the continuous-time setting of the original BSM model, it turns out that such dynamic replication can be made exact by a continuous rebalancing of the hedge portfolio between the stock and cash, such that the amount of stock coincides with the option price sensitivity with respect to the stock price. This makes the total portfolio made of the option and the hedge portfolio *instantaneously risk-free*, or equivalently it makes the option instantaneously perfectly replicable in terms of the stock and cash. Risk of mis-hedging between the option and its underlying is instantaneously eliminated; therefore, the full portfolio involving the option and its hedge should earn a risk-free rate. The option price in this limit does not depend on risk preferences of investors.

Such analysis performed in the continuous-time setting gives rise to the celebrated Black–Scholes partial differential equation (PDE) for option prices, whose solution produces option prices as deterministic functions of current stock prices. The Black–Scholes PDE can be derived using analysis of the hedge portfolio in discrete time with time steps Δt , and then taking the continuous-time limit $\Delta t \rightarrow 0$, see, e.g., Wilmott (1998). It can be shown that the resulting continuous-time BSM model does not amount to a problem of sequential decision making and does not in general reduce to any sort of optimization problem.

However, in option markets, rebalancing of option replication (hedge) portfolio occurs at finite frequency, e.g. daily. A frequent rebalancing can be costly due to transaction costs which are altogether neglected in the classical BSM model. When transaction costs are added, a formal continuous-time limit may not even exist as it leads to formally infinite option prices due to an infinite number of portfolio rebalancing acts.

With a finite rebalancing frequency, perfect replication is no longer feasible, and the replicating portfolio will in general be different from the option value according to the amount of hedge slippage. The latter depends on the stock price evolution between consecutive rebalancing acts for the portfolio. Respectively, in the absence of perfect replication, a hedged option position carries some mis-hedging risk which the option buyer or seller should be compensated for. This means that once we revert from the idealized setting of continuous-time finance to a realistic setting of discrete-time finance, option pricing becomes dependent on investors' risk preferences.

If we take a view of an option seller agent in such a discrete-time setting, its objective should be to minimize some measures of slippage risk, also referred to as a “risk-adjusted cost of hedging” the option, by dynamic option replication. When viewed over the lifetime of an option, this setting can be considered a sequential decision-making process of minimization of slippage cost (or equivalently maximization of rewards determined as negative costs). While such a discrete-time approach converges to the Black–Scholes formulation in the limit of vanishing time steps, it offers both a more realistic setting, and allows one to focus on the

key objective of option trading and pricing, which is *risk minimization by hedging in a sequential decision-making process*. This makes option pricing amenable to methods of reinforcement learning, and indeed, as we will see below, option pricing and hedging in discrete time *amounts* to reinforcement learning.

Casting option pricing as a reinforcement learning task offers a few interesting insights. First, if we select a specific model for the stock price dynamics, we can use model-based reinforcement learning as a powerful sample-based (Monte Carlo) computational approach. The latter may be advantageous to other numerical methods such as finite differences for computing option prices and hedge ratios, especially when dimensionality of the state space goes beyond three or four. Second, we may rely on model-free reinforcement learning methods such as Q-learning, and bypass the need to build a model of stock price dynamics altogether. RL provides a framework for model-free learning of option prices and hedges.¹ While we only consider the simplest setting for a reinforcement learning approach to pricing and hedging of European vanilla options (e.g., put or call options), the approach can be extended in a straightforward manner to more complex instruments including options on multiple assets, early exercises, option portfolios, market frictions, etc.

The model presented in this chapter is referred to as the QLBS model, in recognition of the fact that it combines the Q-learning method of Watkins (1989); Watkins and Dayan (1992) with the method of dynamic option replication of the (time-discretized) Black–Scholes model. As Q-learning is a model-free method, this means that the QLBS model is also model-free. More accurately, it is *distribution-free*: option prices in this approach depend on the chosen utility function, but do not rely on any model for the stock price distribution, and instead use only samples from this distribution.

The QLBS model may also be of interest as a financial model which relates to the literature on hedging and pricing in incomplete markets (Föllmer and Schweizer 1989; Schweizer 1995; Cerný and Kallsen 2007; Potters et al. 2001; Petrelli et al. 2010; Grau 2007). Unlike many previous models of this sort, QLBS ensures a full consistency of hedging and pricing at each time step, all within an efficient and data-driven Q-learning algorithm. Additionally, it *extends* the discrete-time BSM model. Extending Markowitz portfolio theory (Markowitz 1959) to a multi-period setting, Sect. 3 incorporates a drift in a risk/return analysis of the option’s hedge portfolio. This extension allows one to consider both *hedging and speculation* with options in a consistent way within the *same* model, which is a challenge for the standard BSM model or its “phenomenological” generalizations, see, e.g., Wilmott (1998).

Following this approach, it turns out that all results of the classical BSM model (Black and Scholes 1973; Merton 1974) can be obtained as a continuous-time limit $\Delta t \rightarrow 0$ of a multi-period version of the Markowitz portfolio theory (Markowitz 1959), if the dynamics of stock prices are log-normal, and the investment portfolio

¹Here we use the notion of model-free learning in the same context as it is normally used in the machine learning literature, namely as a method that does not rely on an explicit model of feature dynamics. Option prices and hedge ratios in the framework presented in this section depend on a model of rewards, and in this sense are model-dependent.

is self-replicating. However, this limit is *degenerate*: all fluctuations of the “true” option price asymptotically decay in this limit, resulting in a deterministic option price which is independent of risk preferences of an investor. However, as long as the time step Δt is kept finite, both risk of option mis-hedging and dependence of the option price on investor risk preferences persist.

To the extent that option pricing in discrete time amounts to either DP (a.k.a. model-based RL), if a model is known, or RL if a model is unknown, we may say that the classical continuous-time BSM model corresponds to the continuous-time limit of *model-based* reinforcement learning. In such a limit, all data requirements are reduced to just *two* numbers—the current stock price and volatility.

3 Discrete-Time Black–Scholes–Merton Model

We start with a discrete-time version of the BSM model. As is well known, the problem of option hedging and pricing in this formulation amounts to a sequential risk minimization. The main open question is *how* to define risk in an option. In this part, we follow a local risk minimization approach pioneered in the work of Föllmer and Schweizer (1989), Schweizer (1995), Cerný and Kallsen (2007). A similar method was developed by physicists Potters et al. (2001), see also the work by Petrelli et al. (2010). We use a version of this approach suggested in Grau (2007).

In this approach, we take the view of a seller of a European option (e.g., a put option) with maturity T and the terminal payoff of $H_T(S_T)$ at maturity, that depends on a final stock price S_T at that time. To hedge the option, the seller uses proceeds of the sale to set up a replicating (hedge) portfolio Π_t composed of the stock S_t and a risk-free bank deposit B_t . The value of hedge portfolio at any time $t \leq T$ is

$$\Pi_t = u_t S_t + B_t, \quad (10.1)$$

where u_t is a position in the stock at time t , taken to hedge risk in the option.

3.1 Hedge Portfolio Evaluation

As usual, the replicating portfolio tries to exactly match the option price in all possible future states of the world. If we start at maturity T when the option position is closed, the hedge u_t should be closed at the same time, thus we set $u_T = 0$ and therefore

$$\Pi_T = B_T = H_T(S_T), \quad (10.2)$$

which sets a terminal condition for B_T that should hold in all future states of the world at time T .²

To find an amount needed to be held in the bank account at previous times $t < T$, we impose the self-financing constraint which requires that all future changes in the hedge portfolio should be funded from an initially set bank account, without any cash infusions or withdrawals over the lifetime of the option. This implies the following relation that ensures conservation of the portfolio value by a re-hedge at time $t + 1$:

$$u_t S_{t+1} + e^{r \Delta t} B_t = u_{t+1} S_{t+1} + B_{t+1}. \quad (10.3)$$

This relation can be expressed recursively in order to calculate the amount of cash in the bank account to hedge the option at any time $t < T$ using its value at the next time step:

$$B_t = e^{-r \Delta t} [B_{t+1} + (u_{t+1} - u_t) S_{t+1}], \quad t = T - 1, \dots, 0. \quad (10.4)$$

Substituting this into Eq. (10.1) produces a recursive relation for Π_t in terms of its values at later times, which can therefore be solved backward in time, starting from $t = T$ with the terminal condition (10.2), and continued through to the current time $t = 0$:

$$\Pi_t = e^{-r \Delta t} [\Pi_{t+1} - u_t \Delta S_t], \quad \Delta S_t = S_{t+1} - e^{r \Delta t} S_t, \quad t = T - 1, \dots, 0. \quad (10.5)$$

Note that Eqs. (10.4) and (10.5) imply that both B_t and Π_t are not measurable at any $t < T$, as they depend on the future. Respectively, their values today B_0 and Π_0 will be random quantities with some distributions. For any given hedging strategy $\{u_t\}_{t=0}^T$, these distributions can be estimated using Monte Carlo simulation, which first simulates N paths of the underlying $S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_N$, and then evaluates Π_t going backward on each path. Note that because the choice of a hedge strategy does not affect the evolution of the underlying, such simulation of forward paths should only be performed once, and then re-used for future evaluations of the hedge portfolio under different hedge strategy scenarios. Alternatively, the distribution of the hedge portfolio value Π_0 can be estimated using real historical data for stock prices, together with a pre-determined hedging strategy $\{u_t\}_{t=0}^T$ and a terminal condition (10.2).

To summarize, the forward pass of Monte Carlo simulation is done by simulating the process $S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_N$, while the backward pass is performed using the recursion (10.5) which takes a prescribed hedge strategy, $\{u_t\}_{t=0}^T$, and back-propagates uncertainty in the future into uncertainty today, via the self-financing constraint (10.3) (Grau 2007) which serves as a “time machine for risk.”

²When transaction costs are neglected, taking $u_T = 0$ simply means converting all stock into cash. For more details on the choice $u_T = 0$, see Grau (2007).

As a result of such “back-propagation of uncertainty” from the future to the current time t , the option replicating portfolio Π_t at time t is a random quantity with a certain distribution. The option price acceptable to the option seller would then be determined by risk preferences of the option seller. The option price can, for example, be taken to be the mean of the distribution of Π_t , plus some premium for risk. Clearly, the option price can be determined only *after* the seller decides on a *hedging strategy*, $\{u_t\}_{t=0}^T$, to be used in the future, which would be applied in the same way (as a mapping) for any future value, $\{\Pi_t\}_{t=0}^T$. The choice of an *optimal hedge strategy*, $\{u_t\}_{t=0}^T$, will therefore be discussed next.

3.2 Optimal Hedging Strategy

Unlike the recursive calculation of the hedge portfolio value (10.5) which is performed *path-wise*, optimal hedges are computed using a *cross-sectional* analysis that operates simultaneously over all paths. This is because we need to learn a hedging strategy, $\{u_t\}_{t=0}^T$, which would apply to *all* states that might be encountered in the future, but each given path only produces *one* value S_t at time t . Therefore, to compute an optimal hedge, $u_t(S_t)$, for a given time step t , we need *cross-sectional* information on *all* concurrent paths.

As with the portfolio value calculation, the optimal hedges, $\{u_t\}_{t=0}^T$, are computed backward in time, starting from $t = T$. However, because we cannot know the future when we compute a hedge, for each time t , any calculation of an optimal hedge, u_t , can only condition on the information \mathcal{F}_t available at time t . This calculation is similar to the American Monte Carlo method of Longstaff and Schwartz (2001).

► Longstaff–Schwartz American Monte Carlo Option Pricing

While the objective of the American Monte Carlo method of Longstaff and Schwartz (2001) is altogether different from the problem addressed in this chapter (a risk-neutral valuation of an American option vs a real-measure discrete-time hedging/pricing of a European option), the *mathematical* setting is similar. Both problems look for an optimal strategy, and their solution requires a backward recursion in combination with a forward simulation. Here we provide a brief outline of their method.

The main idea of the LSM approach of Longstaff and Schwartz (2001) is to treat the backward-looking stage of the security evaluation as a regression problem formulated in a forward-looking manner which is more suited for

(continued)

a Monte Carlo (MC) setting. The starting point is the (backward-looking) Bellman equation, the most fundamental equation of the stochastic optimal control (otherwise known as stochastic optimization). For an American option on a financial underlying, the control variable is binary: “exercise” or “not exercise.” The Bellman equation for this particular case produces the continuation value, $C_t(S_t)$, at time t as a function of the current underlying value S_t :

$$C_t(S_t) = \mathbb{E} [e^{-r\Delta t} \max(h_{t+\Delta t}(S_{t+\Delta t}), C_{t+\Delta t}(S_{t+\Delta t})) \mid \mathcal{F}_t]. \quad (10.6)$$

Here $h_\tau(S_\tau)$ is the option payoff at time τ . For example, for an American put option, $h_\tau(S_\tau) = (K - S_\tau)_+$. Note that for American options, the continuation value should be estimated as a *function* $C_t(x)$ of the value $x = X_t$, as long we want to know whether it is larger or smaller than the intrinsic value, $H(X_t)$, for a particular *realization* $X_t = x$ of the process X_t at time t . The problem is, of course, that each Monte Carlo path has exactly *one* value of X_t at time t . One way to estimate a function $C_t(S_t)$ is to use all paths, i.e. use the *cross-sectional information*. To this end, the one-step Bellman equation (10.6) is interpreted as a regression of the form

$$\max(h_{t+\Delta t}(S_{t+\Delta t}), C_{t+\Delta t}(S_{t+\Delta t})) = e^{r\Delta t} C_t(S_t) + \varepsilon_t(S_t), \quad (10.7)$$

where $\varepsilon_t(S_t)$ is a random noise at time t with mean zero, which may in general depend on the underlying value S_t at that time. Clearly (10.7) and (10.6) are equivalent in expectations, as taking the expectation of both sides of (10.7), we recover (10.6). Next the unknown function $C_t(S_t)$ is expanded in a set of basis functions:

$$C_t(x) = \sum_n a_n(t) \phi_n(x), \quad (10.8)$$

for some particular choice of the basis $\{\phi_n(x)\}$, and the coefficients $a_n(t)$ are then calculated using the least squares regression of $\max(h_{t+\Delta t}(S_{t+\Delta t}), C_{t+\Delta t}(S_{t+\Delta t}))$ on the value S_t of the underlying at time t across all Monte Carlo paths.

The optimal hedge, $u^*(S_t)$, in this model is obtained from the requirement that the variance of Π_t across all simulated paths at time t is minimized when conditioned on the currently available *cross-sectional* information \mathcal{F}_t , i.e.

$$\begin{aligned} u_t^*(S_t) &= \underset{u}{\operatorname{argmin}} \operatorname{Var} [\Pi_t | \mathcal{F}_t] \\ &= \underset{u}{\operatorname{argmin}} \operatorname{Var} [\Pi_{t+1} - u_t \Delta S_t | \mathcal{F}_t], \quad t = T-1, \dots, 0. \end{aligned} \quad (10.9)$$

Note the first expression in Eq. (10.9) implies that all uncertainty in Π_t is due to uncertainty regarding the amount B_t needed to be held in the bank account at time t in order to meet future obligations at the option maturity T . This means that an optimal hedge should minimize the cost of a hedge capital for the option position at each time step t .

The optimal hedge can be found analytically by setting the derivative of (10.9) to zero. This gives

$$u_t^*(S_t) = \frac{\operatorname{Cov} (\Pi_{t+1}, \Delta S_t | \mathcal{F}_t)}{\operatorname{Var} (\Delta S_t | \mathcal{F}_t)}, \quad t = T-1, \dots, 0. \quad (10.10)$$

This expression involves one-step expectations of quantities at time $t+1$, conditional on time t . How they can be computed depends on whether we deal with a continuous or a discrete state space. If the state space is discrete, then such one-step conditional expectations are simply finite sums involving transition probabilities of an MDP model. If, on the other hand, we work in a continuous-state setting, these conditional expectations can be calculated in a Monte Carlo setting by using expansions in basis functions, similar to the LSMC method of Longstaff and Schwartz (2001), or real-measure MC methods of Grau (2007), Petrelli et al. (2010), Potters et al. (2001).

In our exposition below, we use a general notation as in Eq. (10.10) to denote similar conditional expectations where \mathcal{F}_t denotes cross-sectional information set at time t , which lets us keep the formalism general enough to handle both cases of a continuous and a discrete state spaces, and discuss simplifications that arise in a special case of a discrete-state formulation separately, whenever appropriate.

3.3 Option Pricing in Discrete Time

We start with the notion of a *fair* option price \hat{C}_t defined as a time- t expected value of the hedge portfolio Π_t :

$$\hat{C}_t = \mathbb{E}_t [\Pi_t | \mathcal{F}_t]. \quad (10.11)$$

Using Eq. (10.5) and the tower law of conditional expectations, we obtain

$$\begin{aligned} \hat{C}_t &= \mathbb{E}_t [e^{-r\Delta t} \Pi_{t+1} | \mathcal{F}_t] - u_t(S_t) \mathbb{E}_t [\Delta S_t | \mathcal{F}_t] \\ &= \mathbb{E}_t [e^{-r\Delta t} \mathbb{E}_{t+1} [\Pi_{t+1} | \mathcal{F}_{t+1}] | \mathcal{F}_t] - u_t(S_t) \mathbb{E}_t [\Delta S_t | \mathcal{F}_t] \\ &= \mathbb{E}_t [e^{-r\Delta t} \hat{C}_{t+1} | \mathcal{F}_t] - u_t(S_t) \mathbb{E}_t [\Delta S_t | \mathcal{F}_t], \quad t = T-1, \dots, 0. \end{aligned} \quad (10.12)$$

Note that we can similarly use the tower law of conditional expectations to express the optimal hedge in terms of \hat{C}_{t+1} instead of Π_{t+1} :

$$u_t^*(S_t) = \frac{\text{Cov}(\Pi_{t+1}, \Delta S_t | \mathcal{F}_t)}{\text{Var}(\Delta S_t | \mathcal{F}_t)} = \frac{\text{Cov}\left(\hat{C}_{t+1}, \Delta S_t \middle| \mathcal{F}_t\right)}{\text{Var}(\Delta S_t | \mathcal{F}_t)}. \quad (10.13)$$

If we now substitute (10.13) into (10.12) and re-arrange terms, we can put the recursive relation for \hat{C}_t in the following form:

$$\hat{C}_t = e^{-r\Delta t} \mathbb{E}^{\hat{\mathbb{Q}}} \left[\hat{C}_{t+1} \middle| \mathcal{F}_t \right], \quad t = T - 1, \dots, 0, \quad (10.14)$$

where $\hat{\mathbb{Q}}$ is a signed measure with transition probabilities

$$\tilde{q}(S_{t+1} | S_t) = p(S_{t+1} | S_t) \left[1 - \frac{(\Delta S_t - \mathbb{E}_t[\Delta S_t]) \mathbb{E}_t[\Delta S_t]}{\text{Var}(\Delta S_t | \mathcal{F}_t)} \right], \quad (10.15)$$

and where $p(S_{t+1} | S_t)$ are transition probabilities under the physical measure \mathbb{P} . Note that for sufficiently large moves of ΔS_t , this expression may become negative. This means that $\hat{\mathbb{Q}}$ is not a genuine probability measure, but rather only a *signed* measure (a signed measure, unlike a regular measure, can take both positive and negative values).

A potential for a negative fair option price, \hat{C}_t , is a well-known property of quadratic risk minimization schemes (Cerný and Kallsen 2007; Föllmer and Schweizer 1989; Grau 2007; Potters et al. 2001; Schweizer 1995). However, we note that the “fair” (expected) option price (10.11) is *not* a price a seller of the option should charge. The actual fair *risk-adjusted* price is given by Eq. (10.16) below, which can always be made non-negative by a proper level of risk-aversion λ which is defined by the seller’s risk preferences.³

The reason why the fair option price is not yet the price that the option seller should charge for the option is that she is exposed to the risk of exhausting the bank account B_t at some time in the future, after any fixed amount $\hat{B}_0 = \mathbb{E}_0[B_0]$ is paid into the bank account at time $t = 0$ upon selling the option. If necessary, the option seller would need to add cash to the hedge portfolio and she has to be compensated for such a risk. One possible specification of a risk premium, that the dealer has to add on top of the fair option price, is her own optimal ask price to add to the cumulative expected discounted variance of the hedge portfolio along all time steps $t = 0, \dots, N$, with a risk-aversion parameter λ :

³If it is desired to have non-negative option prices for arbitrary levels of risk-aversion, the method developed below can be generalized by using non-quadratic utility functions instead of the quadratic Markowitz utility. This would incur a moderate computational overhead of numerically solving a convex optimization problem at each time step, instead of a quadratic optimization that is solved semi-analytically.

$$C_0^{(ask)}(S, u) = \mathbb{E}_0 \left[\Pi_0 + \lambda \sum_{t=0}^T e^{-rt} \text{Var} [\Pi_t | \mathcal{F}_t] \middle| S_0 = S, u_0 = u. \right] \quad (10.16)$$

In order to proceed further, we first note that the problem of *minimization* of a fair (to the dealer) option price (10.16) can be equivalently expressed as the problem of *maximization* of its negative $V_t = -C_t^{(ask)}$, where

$$V_t(S_t) = \mathbb{E}_t \left[-\Pi_t - \lambda \sum_{t'=t}^T e^{-r(t'-t)} \text{Var} [\Pi_{t'} | \mathcal{F}_{t'}] \middle| \mathcal{F}_t \right]. \quad (10.17)$$

Example 10.8 Option pricing with non-quadratic utility functions

The fact that the “fair” option price, \hat{C}_t , can become negative when price fluctuations are large is attributed to the non-monotonicity of the Markowitz quadratic utility. Non-monotonicity violates the Von Neumann–Morgenstern conditions $U'(a) \geq 0$, $U''(a) \leq 0$ on a utility function $U(a)$ of a rational investor. While this problem with the quadratic utility function can be resolved by adding a risk premium to the fair option price, this may require that the risk-aversion parameter, λ , exceeds some minimal value. We can obtain non-negative option prices with arbitrary values of risk-aversion if, instead of a quadratic utility, we use utility functions that satisfy the Von Neumann–Morgenstern conditions. In particular, one popular choice is given by the exponential utility function

$$U(X) = -\exp(-\gamma X), \quad (10.18)$$

where γ is a risk-aversion parameter whose meaning is similar to parameter ρ in the quadratic utility. As shown in Halperin (2018), the hedges and prices corresponding to the quadratic risk minimization scheme can be obtained with the exponential utility in the limit of a small risk-aversion $\gamma \rightarrow 0$, alongside calculable corrections via an expansion in powers of γ .

Note that while the idea of adding an option price premium proportional to the variance of the hedge portfolio as done in Eq. (10.16) was initially suggested on the intuitive grounds by Potters et al. (2001), a utility-based approach presented in Halperin (2018) actually *derives* it as a quadratic approximation to a utility-based option price, which also establishes an approximate relation between a risk-aversion parameter λ of the quadratic risk optimization and a parameter γ of the exponential utility $U(X) = -\exp(-\gamma X)$:

$$\lambda \simeq \frac{1}{2}\gamma. \quad (10.19)$$

3.4 Hedging and Pricing in the BS Limit

The framework presented above provides a smooth transition to the strict BS limit $\Delta t \rightarrow 0$. In this limit, the BSM model dynamics under the physical measure \mathbb{P} is described by a continuous-time geometric Brownian motion with a drift, μ , and volatility, σ :

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t, \quad (10.20)$$

where W_t is a standard Brownian motion.

Consider first the optimal hedge strategy (10.13) in the BS limit $\Delta t \rightarrow 0$. Using the first-order Taylor expansion

$$\hat{C}_{t+1} = C_t + \frac{\partial C_t}{\partial S_t} \Delta S_t + O(\Delta t) \quad (10.21)$$

in (10.13), we obtain

$$u_t^{BS}(S_t) = \lim_{\Delta t \rightarrow 0} u_t^*(S_t) = \frac{\partial C_t}{\partial S_t}, \quad (10.22)$$

which is the correct optimal hedge in the continuous-time BSM model.

To find the continuous-time limit of the option price, we first compute the limit of the second term in Eq. (10.12):

$$\lim_{\Delta t \rightarrow 0} u_t(S_t) \mathbb{E}_t [\Delta S_t | \mathcal{F}_t] = \lim_{dt \rightarrow 0} u_t^{BS} S_t (\mu - r) dt = \lim_{dt \rightarrow 0} (\mu - r) S_t \frac{\partial C_t}{\partial S_t} dt. \quad (10.23)$$

To evaluate the first term in Eq. (10.12), we use the second-order Taylor expansion:

$$\begin{aligned} \hat{C}_{t+1} &= C_t + \frac{\partial C_t}{\partial t} dt + \frac{\partial C_t}{\partial S_t} dS_t + \frac{1}{2} \frac{\partial^2 C_t}{\partial S_t^2} (dS_t)^2 + \dots \\ &= C_t + \frac{\partial C_t}{\partial t} dt + \frac{\partial C_t}{\partial S_t} S_t (\mu dt + \sigma dW_t) \\ &\quad + \frac{1}{2} \frac{\partial^2 C_t}{\partial S_t^2} S_t^2 (\sigma^2 dW_t^2 + 2\mu\sigma dW_t dt) + O(dt^2). \end{aligned} \quad (10.24)$$

Substituting Eqs. (10.23) and (10.24) into Eq. (10.12), using $\mathbb{E}[dW_t] = 0$ and $\mathbb{E}[dW_t^2] = dt$, and simplifying, we find that the stock drift μ under the physical measure \mathbb{P} drops out from the problem, and Eq. (10.12) becomes the celebrated Black–Scholes equation in the limit $dt \rightarrow 0$:

$$\frac{\partial C_t}{\partial t} + r S_t \frac{\partial C_t}{\partial S_t} + \frac{1}{2} \sigma^2 S_t^2 \frac{\partial^2 C_t}{\partial S_t^2} - r C_t = 0. \quad (10.25)$$

Therefore, if the stock price is log-normal, both our hedging and pricing formulae become the original formulae of the Black–Scholes–Merton model in the strict limit $\Delta t \rightarrow 0$.

? Multiple Choice Question 1

Select all the following correct statements:

- a. In the Black–Scholes limit $\Delta t \rightarrow 0$, the optimal hedge u_t is equal to the BS delta $\frac{\partial^2 C_t}{\partial S_t^2}$.
 - b. In the Black–Scholes limit $\Delta t \rightarrow 0$, the optimal hedge u_t is equal to the BS delta $\frac{\partial C_t}{\partial S_t}$.
 - c. The risk-aversion parameter λ drops from the problem of option pricing and hedging in the limit $\Delta t \rightarrow 0$.
 - d. For finite Δt , the optimal hedge u_t depends on λ .
-

4 The QLBS Model

We shall now re-formulate and generalize the discrete-time BSM presented in Sect. 3 using the framework of Markov Decision Processes (MDPs). The key idea is that the risk-based pricing and hedging in discrete time can be understood as an MDP problem with the value function to be maximized determined by Eq. (10.17). Recall that we defined this value function as the negative of a risk-adjusted option price for the option seller.

The availability of an MDP formulation for option pricing is beneficial in multiple ways. First, it generalizes the BSM by providing a consistent option pricing and hedging method which can take the expected return of the option into decision making, and thus can be used by both types of market players that trade options: hedgers and speculators. Previous incomplete-market models for option pricing either do not ensure consistency of hedging and pricing, or do not allow for incorporation of stock returns into analysis, or both.⁴ Therefore, the MDP formulation improves the original discrete-time BSM model by making it more generally applicable.

Second, the MDP formulation can be used to formulate new computational approaches to option pricing and hedging. Particular methods are chosen depending on assumptions on a data-generating stock price process. If we assume it is known, so that both transition probabilities and a reward function are *known*, the option

⁴The standard continuous-time BSM model is equivalent to using a risk-neutral pricing measure for option valuation. This approach only enables pure risk-based option hedging, which might be suitable for a hedger but not for an option speculator.

pricing problem can be solved by solving a Bellman optimality equation using dynamic programming or approximate dynamic programming. For the simplest one-stock model formulation, we will show how it can be solved using a combination of Monte Carlo simulation of the underlying process and a recursive semi-analytical procedure that only involves matrix linear algebra (OLS linear regression) for a numerical implementation. Similar methods based on approximate dynamic programming can also be applied to more complex multi-dimensional extensions of the model.

On the other hand, we might know only the general *structure* of an MDP model, but *not* its specifications such as transition probability and reward function. In this case, we should solve a backward recursion for the Bellman optimality equation relying only on *samples* of data. This is the setting of reinforcement learning. It turns out that a Bellman optimality equation for our MDP model, without knowing model dynamics by relying only on *data*, can be easily solved (also semi-analytically, due to a quadratic reward function) using Q-learning or its modifications.

A particular choice between different versions of Q-learning is determined by how the state space is modeled. One can discretize the state and action spaces and work with Markov chain approximation to continuous stock price dynamics, see, e.g., Duan and Simonato (2001). If such a finite-state approximation to dynamics converges to the actual continuous-state dynamics, optimal option prices and hedge ratios computed with this approach also converge to their continuous-state limits. If the log-normal model is indeed the data generation process, prices and hedge ratios convergence to the classical BSM limits, once one further takes the limit $\Delta t \rightarrow 0$, $\lambda \rightarrow 0$ in resulting expressions.

Another possibility is to keep the state space continuous and work with approximate methods to represent a Q-function. In particular, if linear architectures are used, we can use the Fitted Q-iteration (FQI) method (Ernst et al. 2005). For non-linear architectures, neural Q-iteration methods use neural networks to represent a Q-function. Our presentation below is mostly focused on the continuous-state FQI method for the basic single-stock option setting that uses a linear architecture and a fixed set of basis functions. However, all formulas presented below can be easily adjusted to a finite-state formulation by using “one-hot” basis functions.

4.1 State Variables

As stock price dynamics typically involve a deterministic drift term, we can consider a change of state variable such that new, time-transformed variables would be stationary, i.e. non-drifting. For a given stock price process S_t , we can achieve this by defining a new variable X_t by the following relation:

$$X_t = -\left(\mu - \frac{\sigma^2}{2}\right)t + \log S_t. \quad (10.26)$$

The advantage of this representation can be clearly seen in a special case when S_t is a geometric Brownian motion (GBM). For this case, we obtain

$$dX_t = -\left(\mu - \frac{\sigma^2}{2}\right)dt + d\log S_t = \sigma dW_t. \quad (10.27)$$

Therefore, when the true dynamics of S_t is log-normal, X_t is a standard Brownian motion, scaled by volatility, σ . If we know the value of X_t in a given MC scenario, the corresponding value of S_t is given by the formula

$$S_t = e^{X_t + (\mu - \frac{\sigma^2}{2})t}. \quad (10.28)$$

Note that as long as $\{X_t\}_{t=0}^T$ is a martingale, i.e. $\mathbb{E}[dX_t] = 0, \forall t$, on average it should not run too far away from an initial value X_0 during the lifetime of an option. The state variable, X_t , is time-uniform, unlike the stock price, S_t , which has a drift term. But the relation (10.28) can always be used in order to map non-stationary dynamics of S_t onto stationary dynamics of X_t . The martingale property of X_t is also helpful for numerical lattice approximations, as it implies that a lattice should not be too large to capture possible future variations of the stock price.

The change of variables (10.26) and its reverse (10.28) can also be applied when the stock price dynamics are not GBM. Of course, the new state variable X_t will not in general be a martingale in this case; however, it is intrinsically useful for separating non-stationarity of the optimization task from non-stationarity of state variables.

4.2 Bellman Equations

We start by re-stating the risk minimization procedure outlined above in Sect. 3.2 in the language of MDP problems. In particular, time-dependent state variables, S_t , are expressed in terms of time-homogeneous variables X_t using Eq. (10.28). In addition, we will use the notation $a_t = a_t(X_t)$ to denote actions expressed as functions of time-homogeneous variables X_t . Actions, $u_t = u_t(S_t)$, in terms of stock prices are then obtained by the substitution

$$u_t(S_t) = a_t(X_t(S_t)) = a_t\left(\log S_t - \left(\mu - \frac{\sigma^2}{2}\right)t\right), \quad (10.29)$$

where we have used Eq. (10.26).

To differentiate between the actual hedging decisions, $a_t(x_t)$, where x_t is a particular realization of a random state X_t at time t , and a hedging *strategy* that applies for any state X_t , we introduce the notion of a time-dependent *policy*, $\pi(t, X_t)$. We consider deterministic policies of the form:

$$\pi : \{0, \dots, T-1\} \times \mathcal{X} \rightarrow \mathcal{A}, \quad (10.30)$$

which is a deterministic policy that maps the time t and the current state, $X_t = x_t$, to the action $a_t \in \mathcal{A}$:

$$a_t = \pi(t, x_t). \quad (10.31)$$

We start with the value maximization problem of Eq. (10.17), which we rewrite here in terms of a new state variable X_t , and with an upper index to denote its dependence on the policy π :

$$\begin{aligned} V_t^\pi(X_t) &= \mathbb{E}_t \left[-\Pi_t(X_t) - \lambda \sum_{t'=t}^T e^{-r(t'-t)} \text{Var} [\Pi_{t'}(X_{t'}) | \mathcal{F}_{t'}] \middle| \mathcal{F}_t \right] \\ &= \mathbb{E}_t \left[-\Pi_t(X_t) - \lambda \text{Var} [\Pi_t] - \lambda \sum_{t'=t+1}^T e^{-r(t'-t)} \text{Var} [\Pi_{t'}(X_{t'}) | \mathcal{F}_{t'}] \middle| \mathcal{F}_t \right]. \end{aligned} \quad (10.32)$$

The last term in this expression, which involves a sum from $t' = t + 1$ to $t' = T$, can be expressed in terms of V_{t+1} using the definition of the value function with a shifted time argument gives:

$$-\lambda \mathbb{E}_{t+1} \left[\sum_{t'=t+1}^T e^{-r(t'-t)} \text{Var} [\Pi_{t'} | \mathcal{F}_{t'}] \right] = \gamma (V_{t+1} + \mathbb{E}_{t+1} [\Pi_{t+1}]), \quad \gamma := e^{-r\Delta t}. \quad (10.33)$$

Note that parameter γ , introduced in the last relation, is a discrete-time discount factor which in our framework is fixed in terms of a continuous-time risk-free interest rate r of the original BSM model.

Substituting this into (10.32), re-arranging terms and using the portfolio process Eq. (10.5), we obtain the Bellman equation for the QLBS model:

$$V_t^\pi(X_t) = \mathbb{E}_t^\pi [R(X_t, a_t, X_{t+1}) + \gamma V_{t+1}^\pi(X_{t+1})], \quad (10.34)$$

where the one-step time-dependent random reward is defined as follows⁵:

$$\begin{aligned} R_t(X_t, a_t, X_{t+1}) &= \gamma a_t \Delta S_t(X_t, X_{t+1}) - \lambda \text{Var} [\Pi_t | \mathcal{F}_t], \quad t = 0, \dots, T-1 \quad (10.35) \\ &= \gamma a_t \Delta S_t(X_t, X_{t+1}) - \lambda \gamma^2 \mathbb{E}_t \left[\hat{\Pi}_{t+1}^2 - 2a_t \Delta \hat{S}_t \hat{\Pi}_{t+1} + a_t^2 (\Delta \hat{S}_t)^2 \right], \end{aligned}$$

⁵Note that with our definition of the value function Eq. (10.32), it is *not* equal to a discounted sum of future rewards.

where we used Eq. (10.5) in the second line, and $\hat{\Pi}_{t+1} := \Pi_{t+1} - \bar{\Pi}_{t+1}$, and where $\bar{\Pi}_{t+1}$ is the sample mean of all values of Π_{t+1} , and similarly for $\Delta\hat{S}_t$. For $t = T$, we have $R_T = -\lambda \text{Var}[\Pi_T]$, where Π_T is determined by the terminal condition (10.2).

Note that Eq. (10.35) implies that the expected reward, R_t , at time step t is quadratic in the action variable a_t :

$$\begin{aligned}\mathbb{E}_t [R_t(X_t, a_t, X_{t+1})] &= \gamma a_t \mathbb{E}_t [\Delta S_t] \\ &\quad - \lambda \gamma^2 \mathbb{E}_t \left[\hat{\Pi}_{t+1}^2 - 2a_t \Delta \hat{S}_t \hat{\Pi}_{t+1} + a_t^2 (\Delta \hat{S}_t)^2 \right].\end{aligned}\quad (10.36)$$

This expected reward has the same mathematical structure as a risk-adjusted return of a single-period Markowitz portfolio model for a special case of a portfolio made of cash and a single stock. The first term gives the expected return from such portfolio, while the second term penalizes for its quadratic risk. Note further that when $\lambda \rightarrow 0$, the expected reward is *linear* in a_t , so it does not have a maximum.

As the one-step reward in our formulation incorporates variance of the hedge portfolio as a risk penalty, this approach belongs to a class of risk-sensitive reinforcement learning. With our method, risk is incorporated to a traditional risk-neutral RL framework (which only aims at maximization of expected rewards) by modifying the one-step reward function. A similar construction of a risk-sensitive MDP by adding one-step variance penalties to a finite-horizon risk-neutral MDP problem was suggested in a different context by Gosavi (2015).

The action-value function, or Q-function, is defined by an expectation of the same expression as in Eq. (10.32), but conditioned on both the current state X_t and the initial action $a = a_t$, while following a policy π afterwards:

$$\begin{aligned}Q_t^\pi(x, a) &= \mathbb{E}_t [-\Pi_t(X_t) | X_t = x, a_t = a] \\ &\quad - \lambda \mathbb{E}_t^\pi \left[\sum_{t'=t}^T e^{-r(t'-t)} \text{Var} [\Pi_{t'}(X_{t'}) | \mathcal{F}_{t'}] \middle| X_t = x, a_t = a \right].\end{aligned}\quad (10.37)$$

The optimal policy $\pi_t^\star(\cdot | X_t)$ is defined as the policy which maximizes the value function $V_t^\pi(X_t)$, or alternatively and equivalently, maximizes the action-value function $Q_t^\pi(X_t, a_t)$:

$$\pi_t^\star(X_t) = \operatorname{argmax}_\pi V_t^\pi(X_t) = \operatorname{argmax}_{a_t \in \mathcal{A}} Q_t^\star(X_t, a_t). \quad (10.38)$$

The optimal value function satisfies the Bellman optimality equation

$$V_t^\star(X_t) = \mathbb{E}_t^{\pi^\star} [R_t(X_t, u_t = \pi_t^\star(X_t), X_{t+1}) + \gamma V_{t+1}^\star(X_{t+1})]. \quad (10.39)$$

The Bellman optimality equation for the action-value function reads for $t = 0, \dots, T - 1$

$$Q_t^*(x, a) = \mathbb{E}_t \left[R_t(X_t, a_t, X_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(X_{t+1}, a_{t+1}) \mid X_t = x, a_t = a \right], \quad (10.40)$$

with a terminal condition at $t = T$

$$Q_T^*(X_T, a_T = 0) = -\Pi_T(X_T) - \lambda \text{Var}[\Pi_T(X_T)], \quad (10.41)$$

and where Π_T is determined by Eq. (10.2). Recall that $\text{Var}[\cdot]$ here means variance with respect to all Monte Carlo paths that terminate in a given state.

4.3 Optimal Policy

Substituting the expected reward (10.36) into the Bellman optimality equation (10.40), we obtain

$$\begin{aligned} Q_t^*(X_t, a_t) &= \gamma \mathbb{E}_t [Q_{t+1}^*(X_{t+1}, a_{t+1}^*) + a_t \Delta S_t] \\ &\quad - \lambda \gamma^2 \mathbb{E}_t \left[\hat{\Pi}_{t+1}^2 - 2a_t \hat{\Pi}_{t+1} \Delta \hat{S}_t + a_t^2 (\Delta \hat{S}_t)^2 \right], \quad t = 0, \dots, T - 1. \end{aligned} \quad (10.42)$$

Note that the first term $\mathbb{E}_t [Q_{t+1}^*(X_{t+1}, a_{t+1}^*)]$ depends on the current action only through the conditional probability $p(X_{t+1}|X_t a_t)$. However, the next-state probability depends on the current action, a_t , only when there is a feedback loop of trading in the option's underlying stock on the stock price. In the present framework, we follow the standard assumptions of the Black–Scholes model which assumes an option buyer or seller does not produce any market impact.

Neglecting the feedback effect, the expectation $\mathbb{E}_t [Q_{t+1}^*(X_{t+1}, a_{t+1}^*)]$ does not depend on a_t . Therefore, with this approximation, the action-value function $Q_t^*(X_t, a_t)$ is quadratic in the action variable a_t .

> The Black–Scholes Limit

Note that in the limit of zero risk-aversion $\lambda \rightarrow 0$, this equation becomes

$$Q_t^*(X_t, a_t) = \gamma \mathbb{E}_t [Q_{t+1}^*(X_{t+1}, a_{t+1}^*) + a_t \Delta S_t]. \quad (10.43)$$

(continued)

As in this limit $Q_t^*(X_t, a_t) = -\Pi(X_t, a_t)$, using the fair option price definition (10.11), we obtain

$$\hat{C}_t = \gamma \mathbb{E}_t \left[\hat{C}_{t+1} - a_t \Delta S_t \right]. \quad (10.44)$$

This equation coincides with Eq. (10.12), showing that the recursive formula (10.42) correctly rolls back the BSfair option price $\hat{C}_t = \mathbb{E}_t [\Pi_t]$, which corresponds to first taking the limit $\lambda \rightarrow 0$, and then taking the limit $\Delta t \rightarrow 0$ of the QLBS price (while using the BS delta for a_t in Eq. (10.44), see below).

As $Q_t^*(X_t, a_t)$ is a quadratic function of a_t , the optimal action (i.e., the hedge) $a_t^*(S_t)$ that maximizes $Q_t^*(X_t, a_t)$ is computed analytically:

$$a_t^*(X_t) = \frac{\mathbb{E}_t \left[\Delta \hat{S}_t \hat{\Pi}_{t+1} + \frac{1}{2\gamma\lambda} \Delta S_t \right]}{\mathbb{E}_t \left[(\Delta \hat{S}_t)^2 \right]}. \quad (10.45)$$

If we now take the limit of this expression as $\Delta t \rightarrow 0$ by using Taylor expansions around time t as in Sect. 3.4, we obtain (see also Problem 1):

$$\lim_{\Delta t \rightarrow 0} a_t^* = \frac{\partial \hat{C}_t}{\partial S_t} + \frac{\mu - r}{2\lambda\sigma^2} \frac{1}{S_t}. \quad (10.46)$$

Note that if we set $\mu = r$, or alternatively if we take the limit $\lambda \rightarrow \infty$, it becomes identical to the BS delta, while the finite- Δt delta in Eq. (10.45) coincides in these cases with a local risk-minimization delta given by Eq. (10.10). Both these facts have related interpretations. The quadratic hedging that approximates option delta (see Sect. 3.4) only accounts for *risk* of a hedge portfolio, while here we extend it by adding a *drift term* $\mathbb{E}_t [\Pi_t]$ to the objective function, see Eq. (10.17), in the style of Markowitz risk-adjusted portfolio return analysis (Markowitz 1959). This produces a linear first term in the quadratic expected reward (10.36). Resulting hedges are therefore different from hedges obtained by only minimizing risk. Clearly, a pure risk-focused quadratic hedge corresponds to either taking the limit of *infinite* risk-aversion rate in a Markowitz-like risk-return analysis, or setting $\mu = r$ in the above formula, to achieve the same effect. Both factors appearing in Eq. (10.46) show these two possible ways to obtain pure risk-minimizing hedges from our more general hedges. Such hedges can be applied when an option is considered for investment/speculation, rather than only as a hedge instrument.

To summarize, the local risk-minimization hedge and fair price formulae of Sect. 3 are recovered from Eqs. (10.45) and (10.42), respectively, if we first set $\mu = r$ in Eq. (10.45), and then set $\lambda = 0$ in Eq. (10.42). After that, the continuous-time BS formulae for these expressions are reproduced in the final limit $\Delta t \rightarrow 0$ in these resulting expressions, as discussed in Sect. 3. Note that the order of taking the limits is to start with the hedge ratio (10.46), set there $\mu = r$, then substitute this into the price equation (10.42), and take the limit $\lambda \rightarrow 0$ there, leading to Eq. (10.44). The latter relation yields the Black–Scholes equation in the limit $\Delta t \rightarrow 0$ as shown in Eq. (10.25). This order of taking the BS limit is consistent with the principle of hedging first and pricing second, which is implemented in the QLBS model, as well as consistent with market practices of working with illiquid options.

Substituting Eq. (10.45) back into Eq. (10.42), we obtain an explicit recursive formula for the *optimal* action-value function for $t = 0, \dots, T - 1$:

$$Q_t^*(X_t, a_t^*) = \gamma \mathbb{E}_t \left[Q_{t+1}^*(X_{t+1}, a_{t+1}^*) - \lambda \gamma \hat{\Pi}_{t+1}^2 + \lambda \gamma (a_t^*(X_t))^2 (\Delta \hat{S}_t)^2 \right], \quad (10.47)$$

where $a_t^*(X_t)$ is defined in Eq. (10.45). Note that this relation does *not* have the right risk-neutral limit when we set $\lambda \rightarrow 0$ in it. The reason is that setting $\lambda \rightarrow 0$ in Eq. (10.47) is equivalent to setting $\lambda \rightarrow 0$ in Eq. (10.45), but, as we just discussed, this would *not* be the right way to reproduce the BS option price equation (10.25). The correct procedure of taking the limit $\lambda \rightarrow 0$ in the recursion for the Q-function is given by Eq. (10.43) which implies that action a_t used there is obtained as explained above by setting $\mu = r$ in Eq. (10.46).

The backward recursion given by Eqs. (10.45) and (10.47) proceeds all the way backward starting at $t = T - 1$ to the present $t = 0$. At each time step, the problem of maximization over possible actions amounts to convex optimization which is done analytically using Eq. (10.45), which is then substituted into Eq. (10.47) for the current time step. Note that such simplicity of action optimization in the Bellman optimality equation is not encountered very often in other SOC problems. As Eq. (10.47) provides the backward recursion directly for the *optimal* Q-function, neither continuous nor discrete action space representation is required in our setting, as the action in this equation is always just one *optimal* action. If we deal with a finite-state QLBS model, then the values of the optimal time- t Q-function for each node are obtained directly from sums of values of the next-step expectation in various states at time $t + 1$, times one-step probabilities to reach these states.

The end result of the backward recursion for the action-value function is its current value. According to our definition of the option price (10.16), it is exactly the negative of the optimal *Q*-function. We therefore obtain the following expression for the fair ask option price in our approach, which we can refer to as the *QLBS option price*:

$$C_t^{(QLBS)}(S_t, ask) = -Q_t(S_t, a_t^*). \quad (10.48)$$

It is interesting to note that while in the original BSM model the price and the hedge for an option are given by two separate expressions, in the QLBS model, they are parts of the *same* expression (10.48) simply because its option price is the (negative of the) optimal Q-function, whose second argument is by construction the optimal *action*—which corresponds to the optimal hedge in the setting of the QLBS model.

Equations (10.48) and (10.45) that give, respectively, the optimal price and the optimal hedge for the option, jointly provide a complete solution of the QLBS model (when the dynamics are *known*) that generalizes the classical BSM model towards a non-asymptotic case $\Delta t > 0$, while reducing to the latter in the strict BSM limit $\Delta t \rightarrow 0$. In the next section, we will see how they can be implemented.

4.4 DP Solution: Monte Carlo Implementation

In practice, the backward recursion expressed by Eqs. (10.45) and (10.47) is solved in a Monte Carlo setting, where we use N simulated (or real) paths for the state variable X_t . In addition, we assume that we have chosen a set of basis functions $\{\Phi_n(x)\}$.

We can then expand the optimal action (hedge) $a_t^*(X_t)$ and optimal Q-function $Q_t^*(X_t, a_t^*)$ in basis functions, with time-dependent coefficients:

$$a_t^*(X_t) = \sum_n^M \phi_{nt} \Phi_n(X_t), \quad Q_t^*(X_t, a_t^*) = \sum_n^M \omega_{nt} \Phi_n(X_t). \quad (10.49)$$

Coefficients ϕ_{nt} and ω_{nt} are computed recursively backward in time for $t = T - 1, \dots, 0$.

First, we find coefficients ϕ_{nt} of the optimal action expansion. This is found by minimization of the following quadratic functional that is obtained by replacing the expectation in Eq. (10.42) by a MC estimate, dropping all a_t -independent terms, substituting the expansion (10.49) for a_t , and changing the overall sign to convert maximization into minimization:

$$G_t(\phi) = \sum_{k=1}^{N_{MC}} \left(- \sum_n \phi_{nt} \Phi_n(X_t^k) \Delta S_t^k + \gamma \lambda \left(\hat{\Pi}_{t+1}^k - \sum_n \phi_{nt} \Phi_n(X_t^k) \Delta \hat{S}_t^k \right)^2 \right). \quad (10.50)$$

This formulation automatically ensures averaging over market scenarios at time t .

Minimization of Eq. (10.50) with respect to coefficients ϕ_{nt} produces a set of linear equations:

$$\sum_m A_{nm}^{(t)} \phi_{mt} = B_n^{(t)}, \quad n = 1, \dots, M \quad (10.51)$$

where

$$\begin{aligned} A_{nm}^{(t)} &:= \sum_{k=1}^{N_{MC}} \Phi_n\left(X_t^k\right) \Phi_m\left(X_t^k\right) \left(\Delta \hat{S}_t^k\right)^2 \\ B_n^{(t)} &:= \sum_{k=1}^{N_{MC}} \Phi_n\left(X_t^k\right) \left[\hat{\Pi}_{t+1}^k \Delta \hat{S}_t^k + \frac{1}{2\gamma\lambda} \Delta S_t^k \right] \end{aligned} \quad (10.52)$$

which produces the solution for the coefficients of expansions of the optimal action $a_t^*(X_t)$ in a vector form:

$$\boldsymbol{\phi}_t^* = \mathbf{A}_t^{-1} \mathbf{B}_t, \quad (10.53)$$

where \mathbf{A}_t and \mathbf{B}_t are a matrix and vector, respectively, with matrix elements given by Eq. (10.52). Note a similarity between this expression and the general relation (10.45) for the optimal action.

Once the optimal action a_t^* at time t is found in terms of its coefficients (10.53), we turn to the problem of finding coefficients ω_{nt} of the basis function expansion (10.49) for the optimal Q-function. To this end, the one-step Bellman optimality equation (10.40) for $a_t = a_t^*$ is interpreted as regression of the form

$$R_t(X_t, a_t^*, X_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(X_{t+1}, a_{t+1}) = Q_t^*(X_t, a_t^*) + \varepsilon_t, \quad (10.54)$$

where ε_t is a random noise at time t with mean zero. Clearly, taking expectations of both sides of (10.54), we recover Eq. (10.40) with $a_t = a_t^*$; therefore, Eqs. (10.54) and (10.40) are equivalent in expectations when $a_t = a_t^*$.

Coefficients ω_{nt} are therefore found by solving the following least square optimization problem:

$$F_t(\omega) = \sum_{k=1}^{N_{MC}} \left(R_t(X_t, a_t^*, X_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(X_{t+1}, a_{t+1}) - \sum_n \omega_{nt} \Phi_n(X_t^k) \right)^2. \quad (10.55)$$

Introducing another pair of a matrix \mathbf{C}_t and a vector \mathbf{D}_t with elements

$$\begin{aligned} C_{nm}^{(t)} &:= \sum_{k=1}^{N_{MC}} \Phi_n\left(X_t^k\right) \Phi_m\left(X_t^k\right) \\ D_n^{(t)} &:= \sum_{k=1}^{N_{MC}} \Phi_n\left(X_t^k\right) \left(R_t(X_t, a_t^*, X_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(X_{t+1}, a_{t+1}) \right). \end{aligned} \quad (10.56)$$

We obtain the vector-valued solution for optimal weights ω_t defining the optimal Q-function at time t :

$$\omega_t^* = \mathbf{C}_t^{-1} \mathbf{D}_t. \quad (10.57)$$

Equations (10.53) and (10.57) computed jointly and recursively for $t = T-1, \dots, 0$ provide a practical implementation of the backward recursion scheme of Sect. 4.3 in a continuous-space setting using expansions in basis functions. This approach can be used to find optimal price and optimal hedge when the dynamics are known.

? Multiple Choice Question 2

Select all the following correct statements:

- a. The coefficients of expansion of the Q-function in the QLBS model are obtained in the DP solution from the Bellman equation interpreted as a classification problem, which is solved using deep learning.
 - b. The coefficients of expansion of the Q-function in the QLBS model are obtained in the DP solution from the Bellman equation interpreted as a regression problem, which is solved using least square minimization.
 - c. The DP solution requires rewards to be observable.
 - d. The DP solution computes rewards as a part of the hedge optimization.
-

4.5 RL Solution for QLBS: Fitted Q Iteration

When the transition probabilities and reward functions are not known, the QLBS model can be solved using reinforcement learning. In this section, we demonstrate this approach using a version of Q-learning that is formulated for continuous state-action spaces and is known as Fitted Q Iteration.

Our setting assumes a *batch-mode* learning, when we only have access to some historically collected data. The data available is given by a set of N_{MC} trajectories for the underlying stock S_t (expressed as a function of X_t using Eq. (10.26)), hedge position a_t , instantaneous reward R_t , and the next-time value X_{t+1} :

$$\mathcal{F}_t^{(n)} = \left\{ \left(X_t^{(n)}, a_t^{(n)}, R_t^{(n)}, X_{t+1}^{(n)} \right) \right\}_{t=0}^{T-1}, \quad n = 1, \dots, N_{MC}. \quad (10.58)$$

We assume that such dataset is available either as a simulated data, or as a real historical stock price data, combined with real trading data or artificial data that would track the performance of a hypothetical stock-and-cash replicating portfolio for a given option.

A starting point of the Fitted Q Iteration (FQI) (Ernst et al. 2005; Murphy 2005) method is a choice of a parametric family of models for quantities of interest, namely optimal action and optimal action-value function. We use linear architectures where functions sought are *linear* in adjustable parameters that are next optimized to find the optimal action and action-value function.

We use the same set of basis functions $\{\Phi_n(x)\}$ as we used above in Sect. 4.4. As the optimal Q-function $Q_t^*(X_t, a_t)$ is a quadratic function of a_t , we can represent it as an expansion in basis functions, with time-dependent coefficients parameterized by a matrix \mathbf{W}_t :

$$\begin{aligned} Q_t^*(X_t, a_t) &= \left(1, a_t, \frac{1}{2}a_t^2\right) \begin{pmatrix} W_{11}(t) & W_{12}(t) & \cdots & W_{1M}(t) \\ W_{21}(t) & W_{22}(t) & \cdots & W_{2M}(t) \\ W_{31}(t) & W_{32}(t) & \cdots & W_{3M}(t) \end{pmatrix} \begin{pmatrix} \Phi_1(X_t) \\ \vdots \\ \Phi_M(X_t) \end{pmatrix} \\ &:= \mathbf{A}_t^T \mathbf{W}_t \Phi(X_t) := \mathbf{A}_t^T \mathbf{U}_W(t, X_t). \end{aligned} \quad (10.59)$$

Equation (10.59) is further re-arranged to convert it into a product of a parameter vector and a vector that depends on both the state and the action:

$$\begin{aligned} Q_t^*(X_t, a_t) &= \mathbf{A}_t^T \mathbf{W}_t \Phi(X) = \sum_{i=1}^3 \sum_{j=1}^M \left(\mathbf{W}_t \odot (\mathbf{A}_t \otimes \Phi^T(X)) \right)_{ij} \\ &= \mathbf{W}_t \cdot \text{vec}(\mathbf{A}_t \otimes \Phi^T(X)) := \mathbf{W}_t \Psi(X_t, a_t). \end{aligned} \quad (10.60)$$

Here \odot and \otimes stand for an element-wise (Hadamard) and outer (Kronecker) product of two matrices, respectively. The vector of time-dependent parameters \mathbf{W}_t is obtained by concatenating columns of matrix \mathbf{W}_t , and similarly, $\Psi(X_t, a_t) = \text{vec}(\mathbf{A}_t \otimes \Phi^T(X))$ denotes a vector obtained by concatenating columns of the outer product of vectors \mathbf{A}_t and $\Phi(X)$.

Coefficients \mathbf{W}_t can now be computed recursively backward in time for $t = T - 1, \dots, 0$. To this end, the one-step Bellman optimality equation (10.40) is interpreted as regression of the form

$$R_t(X_t, a_t, X_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(X_{t+1}, a_{t+1}) = \mathbf{W}_t \Psi(X_t, a_t) + \varepsilon_t, \quad (10.61)$$

where ε_t is a random noise at time t with mean zero. Equations (10.61) and (10.40) are equivalent in expectations, as taking the expectation of both sides of (10.61), we recover (10.40) with function approximation (10.59) used for the optimal Q-function $Q_t^*(x, a)$.

Coefficients \mathbf{W}_t are therefore found by solving the following least square optimization problem:

$$\mathcal{L}_t(\mathbf{W}_t) = \sum_{k=1}^{N_{MC}} \left(R_t(X_t, a_t, X_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^*(X_{t+1}, a_{t+1}) - \mathbf{W}_t \Psi(X_t, a_t) \right)^2. \quad (10.62)$$

Note that this relation holds for a general *off-model, off-policy* setting of the Fitted Q Iteration method of RL.

Performing minimization, we obtain

$$W_t^* = S_t^{-1} \mathbf{M}_t, \quad (10.63)$$

where

$$S_{nm}^{(t)} := \sum_{k=1}^{N_{MC}} \Psi_n \left(X_t^k, a_t^k \right) \Psi_m \left(X_t^k, a_t^k \right) \quad (10.64)$$

$$M_n^{(t)} := \sum_{k=1}^{N_{MC}} \Psi_n \left(X_t^k, a_t^k \right) \left(R_t \left(X_t^k, a_t^k, X_{t+1}^k \right) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_{t+1}^* \left(X_{t+1}^k, a_{t+1} \right) \right)$$

To perform the maximization step in the second equation in (10.64) analytically, note that because coefficients \mathbf{W}_{t+1} and hence vectors $\mathbf{U}_W(t+1, X_{t+1}) := \mathbf{W}_{t+1} \Phi(X_{t+1})$ (see Eq. (10.59)) are known from the previous step, we have

$$\begin{aligned} Q_{t+1}^*(X_{t+1}, a_{t+1}^*) &= \mathbf{U}_W^{(0)}(t+1, X_{t+1}) + a_{t+1}^* \mathbf{U}_W^{(1)}(t+1, X_{t+1}) \\ &\quad + \frac{(a_{t+1}^*)^2}{2} \mathbf{U}_W^{(2)}(t+1, X_{t+1}). \end{aligned} \quad (10.65)$$

We emphasize here that while this is a quadratic expression in a_{t+1}^* , it would be *wrong* to use a point of its maximum as a function of a_{t+1}^* as such an optimal value in Eq. (10.65). This would amount to using the same dataset to estimate both the optimal action and the optimal Q-function, leading to an overestimation of $Q_{t+1}^*(X_{t+1}, a_{t+1}^*)$ in Eq. (10.64), due to Jensen's inequality and convexity of the $\max(\cdot)$ function. The correct approach for using Eq. (10.65) is to input a value of a_{t+1}^* computed using the analytical solution Eq. (10.45) (implemented in the sample-based approach in Eq. (10.53)), applied at the previous time step. Due to the availability of the analytical optimal action (10.45), a potential overestimation problem—a classical problem of Q-learning that is sometimes addressed using such methods as Double Q-learning (van Hasselt 2010)—is avoided in the QLBS model, leading to numerically stable results.

Equation (10.63) gives the solution for the QLBS model in a *model-free* and *off-policy* setting, via its reliance on Fitted Q Iteration which is a model-free and off-policy algorithm (Ernst et al. 2005; Murphy 2005).

? Multiple Choice Question 3

Select all the following correct statements:

- Unlike the classical Black–Scholes model, the discrete-time QLBS model explicitly prices mis-hedging risk of the option because it maximizes the Q-function which incorporates mis-hedging risk as a penalty.
 - Counting by the number of parameters to learn, the RL setting for the QLBS model has more unknowns, but also a higher dimensionality of data (more features per observation) than the DP setting.
 - The BS solution is recovered from the RL solution in the limit $\Delta t \rightarrow 0$ and $\lambda \rightarrow 0$.
 - The RL solution is recovered from the BS solution in the limit $\Delta t \rightarrow \infty$ and $\lambda \rightarrow \infty$.
-

4.6 Examples

Here we illustrate the performance of the QLBS model using simulated stock price histories S_t with the initial stock price $S_0 = 100$, stock drift $\mu = 0.05$, and volatility $\sigma = 0.15$. Option maturity is $T = 1$ year, and a risk-free rate is $r = 0.03$. We consider an ATM (“at-the-money”) European put option with strike $K = 100$. Re-hedges are performed bi-weekly (i.e., $\Delta t = 1/24$). We use $N = 50,000$ Monte Carlo scenarios of the stock price trajectory and report results obtained with two MC runs (each having N paths), where the error reported is equal to one standard deviation calculated from these runs. In our experiments, we use pure risk-based hedges, i.e. omit the second term in the numerator in Eq.(10.45), for ease of comparison with the BSM model.

We use 12 basis functions chosen to be cubic B-splines on a range of values of X_t between the smallest and largest values observed in a dataset.

In our experiments below, we pick the Markowitz risk-aversion parameter $\lambda = 0.001$. This provides a visible difference of QLBS prices from BS prices, while being not too far away from BS prices. The dependence of the ATM option price on λ is shown in Fig. 10.1.

Simulated path and solutions for optimal hedges, portfolio values, and Q-function values corresponding to the DP solution of Sect. 4.4 are illustrated in Fig. 10.2.

The resulting QLBS ATM put option price is 4.90 ± 0.12 (based on two MC runs), while the BS price is 4.53.

We first report results obtained with *on-policy* learning with $\lambda = 0.001$. In this case, optimal actions and rewards computed as a part of a DP solution are used as inputs to the Fitted Q Iteration algorithm of Sect. 4.5 and the IRL method of Sect. 10.2, in addition to the paths of the underlying stock. Results of two MC

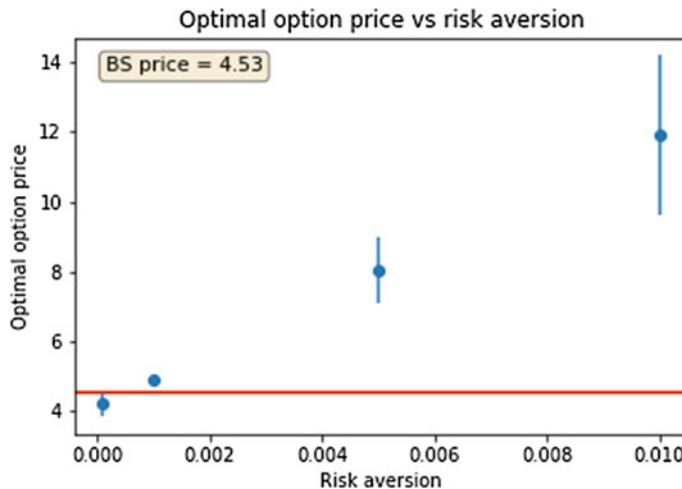


Fig. 10.1 The ATM put option price vs risk-aversion parameter. The time step is $\Delta t = 1/24$. The horizontal red line corresponds to the continuous-time BS model price. Error bars correspond to one standard deviation of two MC runs

batches with Fitted Q Iteration algorithm of Sect. 4.5 are shown (respectively, in the left and right columns, with a random selection of a few trajectories) in Fig. 10.3. Similar to the DP solution, we add a unit matrix with a regularization parameter of 10^{-3} to invert matrix C_t in Eq. (10.63). Note that because here we use *on-policy* learning, the resulting optimal Q-function $Q_t^*(X_t, a_t)$ and its optimal value $Q_t^*(X_t, a_t^*)$ are virtually identical in the graph. The resulting QLBS RL put price is 4.90 ± 0.12 which is identical to the DP value. As expected, the IRL method of Sect. 10.2 produces the same result.

In the next set of experiments we consider *off-policy* learning. The risk-aversion parameter is $\lambda = 0.001$. To generate off-policy data, we multiply, at each time step, optimal hedges computed by the DP solution of the model by a random uniform number in the interval $[1 - \eta, 1 + \eta]$, where $0 < \eta < 1$ is a parameter controlling the noise level in the data. We will consider the values of $\eta = [0.15, 0.25, 0.35, 0.5]$ to test the noise tolerance of our algorithms. Rewards corresponding to these sub-optimal actions are obtained using Eq. (10.35). In Fig. 10.4 we show results obtained for off-policy learning with 10 different scenarios of sub-optimal actions obtained by random perturbations of a fixed simulated dataset. Note that the impact of sub-optimality of actions in recorded data is rather mild, at least for a moderate level of noise. This is as expected as long as Fitted Q Iteration is an *off-policy* algorithm. This implies that when dataset is large enough, the QLBS model can learn even from data with purely random actions. In particular, if the stock prices are log-normal, it can learn the BSM model itself.

Results of two MC batches for off-policy learning with the noise parameter $\eta = 0.5$ with Fitted Q Iteration algorithm are shown in Fig. 10.5.

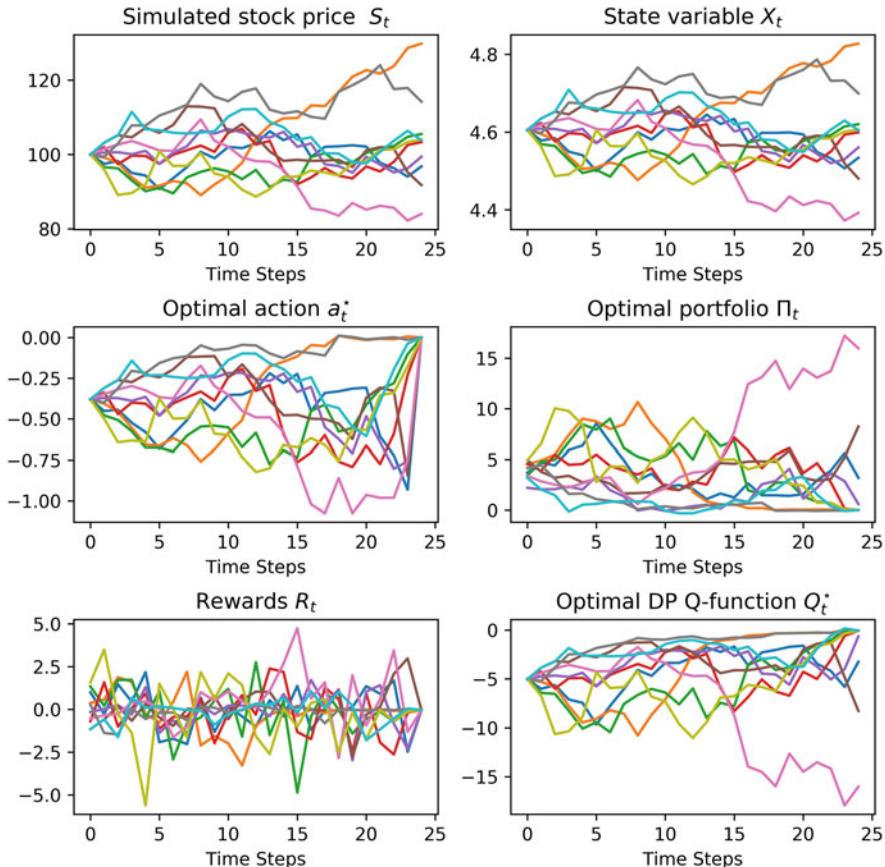


Fig. 10.2 The DP solution for the ATM put option on a subset of MC paths

4.7 Option Portfolios

Thus far we have only considered the problem of hedging and pricing of a single European option by an option seller that *does not* have any pre-existing option portfolio. Here we outline a simple generalization to the case when the option seller *does* have such a pre-existing option portfolio, or alternatively if she seeks to sell a few options simultaneously.

In this case, she is concerned with *consistency* of pricing and hedging of *all* options in her new portfolio. In other words, she has to solve the notorious *volatility smile problem* for her particular portfolio. Here we outline how she can solve it using the QLBS model, illustrating the flexibility and data-driven nature of the model. Such flexibility facilitates adaptation to arbitrary consistent volatility surfaces.

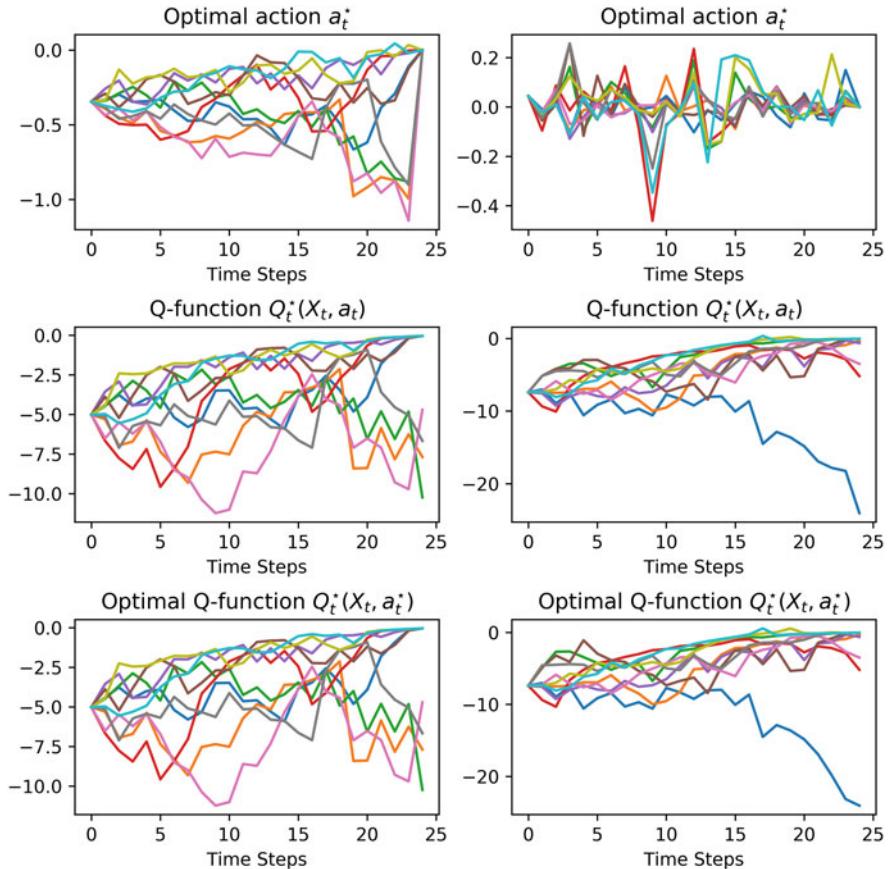


Fig. 10.3 The RL solution (Fitted Q Iteration) for *on-policy* learning for the ATM put option on a subset of MC paths for two MC batches

Assume the option seller has a pre-existing portfolio of K options with market prices C_1, \dots, C_K . All these options reference an underlying state vector (market) \mathbf{X}_t which can be high-dimensional such that each particular option C_i with $i = 1, \dots, K$ references only one or a few components of market state \mathbf{X}_t .

Alternatively, we can add vanilla option prices as components of the market state \mathbf{X}_t . In this case, our dynamic replicating portfolio would include vanilla options, along with underlying stocks. Such hedging portfolio would provide a dynamic generalization of static option hedging for exotics introduced by Carr et al. (1988).

We assume that we have a historical dataset \mathcal{F} which includes N observations of trajectories of tuples of vector-valued market factors, actions (hedges), and rewards (compare with Eq. (10.58)):

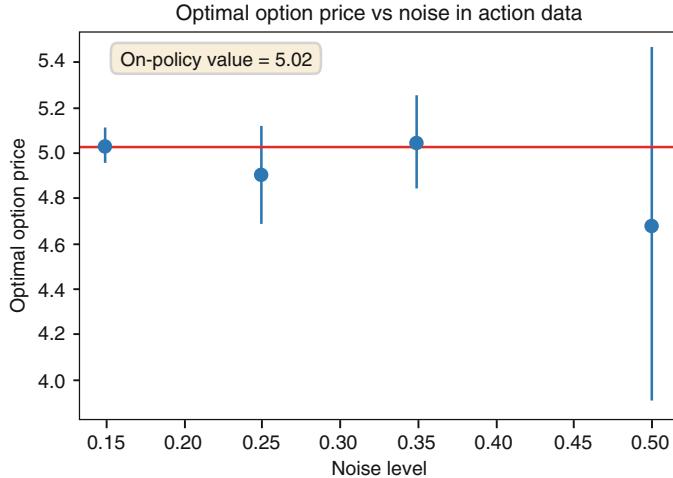


Fig. 10.4 Means and standard deviations of option prices obtained with *off-policy* FQI learning with data obtained by randomization of DP optimal actions by multiplying each optimal action by a uniform random variable in the interval $[1 - \eta, 1 + \eta]$ for $\eta = [0.15, 0.25, 0.35, 0.5]$. Error bars are obtained with 10 scenarios for each value of η . The horizontal red line shows the value obtained with *on-policy* learning corresponding to $\eta = 0$

$$\mathcal{F}_t^{(n)} = \left\{ \left(\mathbf{X}_t^{(n)}, \mathbf{a}_t^{(n)}, \mathbf{R}_t^{(n)}, \mathbf{X}_{t+1}^{(n)} \right) \right\}_{t=0}^{T-1}, \quad n = 1, \dots, N. \quad (10.66)$$

We now assume that the option seller seeks to add to this pre-existing portfolio another (exotic) option C_e (or alternatively, she seeks to sell a portfolio of options C_1, \dots, C_K, C_e). Depending on whether the exotic option C_e was traded before in the market or not, there are two possible scenarios. We shall analyze these scenarios one by one.

In the first case, the exotic option C_e was previously traded in the market (by the seller herself, or by someone else). As long as its deltas and related P&L impacts marked by a trading desk are available, we can simply extend vectors of actions $\mathbf{a}_t^{(n)}$ and rewards $\mathbf{R}_t^{(n)}$ in Eq. (10.66), and then proceed with the FQI algorithm of Sect. 4.5 (or with the IRL algorithm of Sect. 10.2, if rewards are not available). The outputs of the algorithm will be the optimal price P_t of the whole option portfolio, plus optimal hedges for all options in the portfolio. Note that as long as FQI is an *off-policy* algorithm, it is quite forgiving to human or model errors: deltas in the data should not even be perfectly mutually consistent (see single-option examples in the previous section). But of course, the more consistency in the data, the less data is needed to learn an optimal portfolio price P_t .

Once the optimal time-zero value P_0 of the total portfolio C_1, \dots, C_K, C_e is computed, a market-consistent price for the exotic option is simply given by a subtraction:

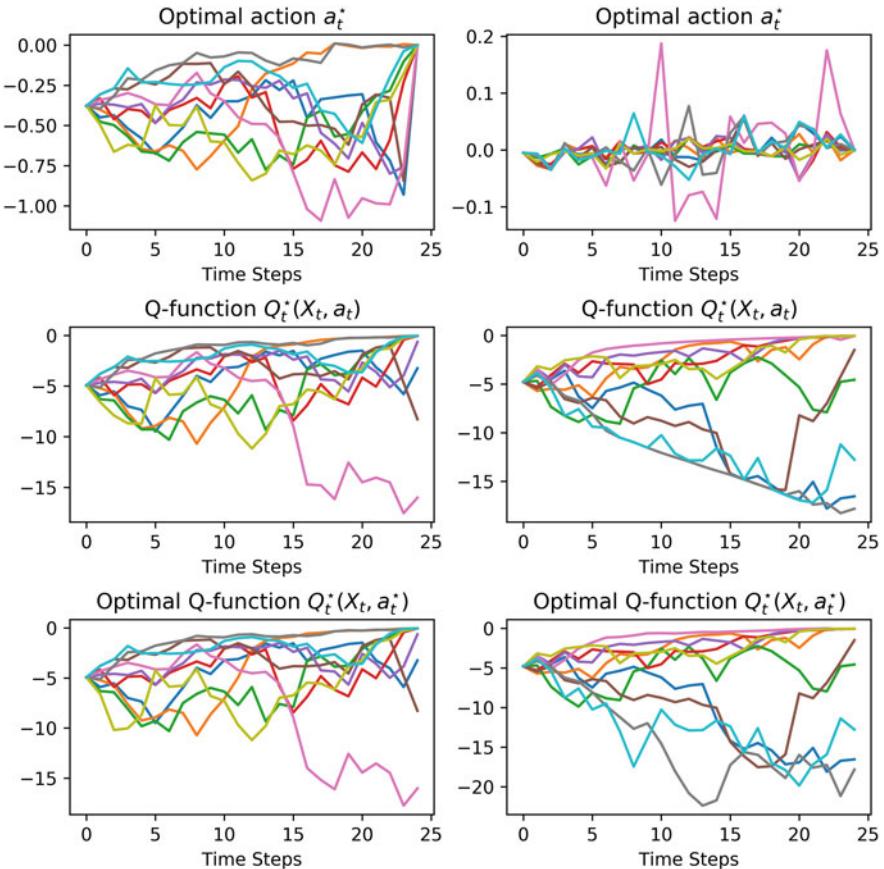


Fig. 10.5 The RL solution (Fitted Q Iteration) for *off-policy* learning with noise parameter $\eta = 0.5$ for the ATM put option on a subset of MC paths for two MC batches

$$C_e = P_0 - \sum_{i=1}^K C_i. \quad (10.67)$$

Note that by construction, the price C_e is consistent with *all* option prices C_1, \dots, C_K and *all* their hedges, to the extent they are consistent between themselves (again, this is because Q-learning is an off-policy algorithm).

Now consider a different case, when the exotic option C_e was *not* previously traded in the market, and therefore there are no available historical hedges for this option. This can be handled by the QLBS model in essentially the same way as in the previous case. Again, because Q-learning is an *off-policy* algorithm, it means that a delta and a reward of a proxy option C_e (that *was* traded before) to C_e could be used in the scheme just described in lieu of their actual values for option C_e .

Consistent with common intuition, this will just slow down the learning, so that more data would be needed to compute the optimal price and hedge for the exotic C_e . On the other hand, the closer the traded proxy C'_e to the actual exotic C_e the option seller wants to hedge and price, the more it helps the algorithm on the data demand side.

4.8 Possible Extensions

So far we have presented the QLBS model in its most basic setting where it is applied to a single European vanilla option such as a put or call option. This framework can be extended or generalized along several directions. Here we overview them, in the order of increasing complexity of changes that would be needed on top of the basic computational framework presented above.

The simplest extension of the basic QLBS setting is to apply it to European options with a non-vanilla terminal payoff, e.g. to a straddle option. Clearly, the only change needed to the basic QLBS setting in this case would be a different terminal condition for the action-value function.

The second extension that is easy to incorporate in the basic QLBS setting are early exercise features for options. This can be added to the QLBS model in much the same way as they are implemented in the American Monte Carlo method of Longstaff and Schwartz. Namely, in the backward recursion, at each time step where an early option exercise is possible, the optimal action-value function is obtained by comparing its value from the next time step continued to the current time step with an intrinsic option value. The latter is defined as a payoff from an immediate exercise of the option, see also Exercise 10.2.

One more possible extension involves capturing higher moments of the replicating portfolio. This assumes using a non-quadratic utility function. One approach is to use an exponential utility function as was outlined above (see also in Halperin (2018)). On the computational side, using a non-quadratic utility gives rise to the need to solve a convex optimization problem at each time step, instead of quadratic optimization.

The basic QLBS framework can also be extended by incorporating transaction costs. This requires re-defining the state and action spaces in the problem. As in the presence of transaction costs holding cash is not equivalent to holding a stock, for this case we can use changes in the stock holding as action variables, while the current stock holding and the stock market price should now be made parts of a state vector. Depending on a functional model for transaction cost, the resulting optimization problem can be either quadratic (if both the reward and transaction cost functions are quadratic in action), or convex, if both these functions are convex.

Finally, the basic framework can be generalized to a multi-asset setting, including option portfolios. The main challenge of such task would be to specify a good set of basis functions. In multiple dimensions, this might be a challenging problem. Indeed, a simple method to form a basis in a multi-dimensional space is to take a

direct (cross) product of individual bases, but this produces an exponential number of basis functions. As a result, such a naive approach becomes intractable beyond a rather low (< 10) number of dimensions.

Feature selection in high-dimensional spaces is a general problem in machine learning, which is not specific to reinforcement learning or the QLBS approach. The latter can benefit from methods developed in the literature. Rather than pursuing this direction, we now turn to a different and equally canonical finance application, namely the multi-period optimization of stock portfolios. We will show that such a multi-asset setting may entirely avoid the need to choose basis functions.

5 G-Learning for Stock Portfolios

5.1 *Introduction*

In this section, we consider a multi-dimensional setting with a multi-asset investment portfolio. Specifically, we consider a stock portfolio, although similar methods can be used for portfolios of other assets, including options.

As we mentioned above in this chapter, one challenge with scaling to multiple dimensions with reinforcement learning is the computational cost and the problem of under-sampling due to the curse of dimensionality. Another potential (and related) issue is the pronounced importance of noise in data. With finite samples, estimations of functions such as the action-value function or the policy function with noisy high-dimensional data can become quite noisy themselves. Rather than relying on deterministic policies as in Q-learning, we may prefer to work with probabilistic methods where such noise can be captured.

A framework that is presented below is designed as a probabilistic approach that scales to a very high-dimensional setting. Again, for ease of exposition, we consider methods for quadratic (Markowitz) reward functions; however, the approach can be generalized to include other reward (utility) functions.

Our approach is based on a probabilistic extension of Q-learning known in the literature as “G-learning.” While G-learning was initially formulated for finite MDPs, here we extend it to a continuous-state and continuous-action case. For an arbitrary reward function, this requires relying on a set of pre-specified basis functions, or using universal function approximators (e.g., neural networks) to represent the action-value function. However, as we will see below, when a reward function is quadratic, neither approach is needed, and the portfolio optimization procedure is semi-analytic.

5.2 Investment Portfolio

We adopt the notation and assumption of the portfolio model suggested by Boyd et al. (2017). In this model, dollar values of positions in n assets $i = 1, \dots, n$ are denoted as a vector \mathbf{x}_t with components $(x_t)_i$ for a dollar value of asset i at the beginning of period t . In addition to assets \mathbf{x}_t , an investment portfolio includes a risk-free bank cash account b_t with a risk-free interest rate r_f . A short position in any asset i then corresponds to a negative value $(x_t)_i < 0$. The vector of mean of bid and ask prices of assets at the beginning of period t is denoted as \mathbf{p}_t , with $(p_t)_i > 0$ being the price of asset i . Trades \mathbf{u}_t are made at the beginning of interval t , so that asset values \mathbf{x}_t^+ immediately after trades are deterministic:

$$\mathbf{x}_t^+ = \mathbf{x}_t + \mathbf{u}_t. \quad (10.68)$$

The total portfolio value is

$$v_t = \mathbb{1}^T \mathbf{x}_t + b_t, \quad (10.69)$$

where $\mathbb{1}$ is a vector of ones. The post-trade portfolio is therefore

$$v_t^+ = \mathbb{1}^T \mathbf{x}_t^+ + b_t^+ = \mathbb{1}^T (\mathbf{x}_t + \mathbf{u}_t) + b_t^+ = v_t + \mathbb{1}^T \mathbf{u}_t + b_t^+ - b_t. \quad (10.70)$$

We assume that all rebalancing of stock positions are financed from the bank cash account (additional cash costs related to the trade will be introduced below). This imposes the following “self-financing” constraint:

$$\mathbb{1}^T \mathbf{u}_t + b_t^+ - b_t = 0, \quad (10.71)$$

which simply means that the portfolio value remains unchanged upon an instantaneous rebalancing of the wealth between the stock and cash:

$$v_t^+ = v_t. \quad (10.72)$$

The post-trade portfolio, v_t^+ and cash are invested at the beginning of period t until the beginning of the next period. The return of asset i over period t is defined as

$$(r_t)_i = \frac{(p_{t+1})_i - (p_t)_i}{(p_t)_i}, \quad i = 1, \dots, n. \quad (10.73)$$

Asset positions at the next time period are then given by

$$\mathbf{x}_{t+1} = \mathbf{x}_t^+ + \mathbf{r}_t \circ \mathbf{x}_t^+, \quad (10.74)$$

where \circ denotes an element-wise (Hadamard) product, and $\mathbf{r}_t \in \mathbb{R}^n$ is the vector of asset returns from period t to period $t + 1$. The next-period portfolio value is then obtained as follows:

$$v_{t+1} = \mathbb{1}^T \mathbf{x}_{t+1} \quad (10.75)$$

$$= (1 + \mathbf{r}_t)^T \mathbf{x}_t^+ \quad (10.76)$$

$$= (1 + \mathbf{r}_t)^T (\mathbf{x}_t + \mathbf{u}_t). \quad (10.77)$$

Given a vector of returns \mathbf{r}_t in period t , the change of the portfolio value in excess of a risk-free growth is

$$\begin{aligned} \Delta v_t := v_{t+1} - (1 + r_f)v_t &= (\mathbb{1} + \mathbf{r}_t)^T (\mathbf{x}_t + \mathbf{u}_t) + (1 + r_f)b_t^+ \\ &\quad - (1 + r_f)\mathbb{1}^T \mathbf{x}_t - (1 + r_f)b_t \\ &= (\mathbf{r}_t - r_f\mathbb{1})^T (\mathbf{x}_t + \mathbf{u}_t), \end{aligned} \quad (10.78)$$

where in the second equation we used Eq. (10.71).

5.3 Terminal Condition

As we generally assume a finite-horizon portfolio optimization with a finite investment horizon T , we must supplement the problem with a proper terminal condition at time T .

For example, if the investment portfolio should track a given benchmark portfolio (e.g., the market portfolio), a terminal condition is obtained from the requirement that at time T , all stock positions should be equal to the actual observed weights of stocks in the benchmark \mathbf{x}_T^B . This implies that $\mathbf{x}_T = \mathbf{x}_T^B$. By Eq. (10.68), this fixes the action \mathbf{u}_T at the last time step:

$$\mathbf{u}_T = \mathbf{x}_T^M - \mathbf{x}_{T-1}. \quad (10.79)$$

Therefore, action \mathbf{u}_T at the last step is deterministic and is not subject to optimization that should be applied to T remaining actions $\mathbf{u}_{T-1}, \dots, \mathbf{u}_0$.

Alternatively, the goal of the investment portfolio can be maximization of a risk-adjusted cumulative reward of the portfolio. In this case, an appropriate terminal condition could be $\mathbf{x}_T = 0$, meaning that any remaining long stock positions should be converted to cash at time T .

5.4 Asset Returns Model

We assume the following linear specification of one-period excess asset returns:

$$\mathbf{r}_t - r_f \mathbb{1} = \mathbf{W}\mathbf{z}_t - \mathbf{M}^T \mathbf{u}_t + \varepsilon_t, \quad (10.80)$$

where \mathbf{z}_t is a vector of predictors with factor loading matrix \mathbf{W} , \mathbf{M} is a matrix of permanent market impacts with a linear impact specification, and ε_t is a vector of residuals with

$$\mathbb{E}[\varepsilon_t] = 0, \quad \mathbb{V}[\varepsilon_t] = \Sigma_r, \quad (10.81)$$

where $\mathbb{E}[\cdot]$ denotes an expectation with respect to the physical measure \mathbb{P} .

Equation (10.80) specifies stochastic returns \mathbf{r}_t , or equivalently the next-step stock prices, as driven by external signals \mathbf{z}_t , control (action) variables \mathbf{u}_t , and uncontrollable noise ε_t .

Though they enter “symmetrically” in Eq. (10.80), two drivers of returns \mathbf{z}_t and \mathbf{u}_t play entirely different roles. While signals \mathbf{z}_t are completely *external* for the agent, actions \mathbf{u}_t are *controlled* degrees of freedom. In our approach, we will be looking for *optimal* controls \mathbf{u}_t for the market-wise portfolio. When we set up a proper optimization problem, we solve for an optimal action \mathbf{u}_t . As will be shown in this section, this optimal control turns out to be a linear function of \mathbf{x}_t , plus noise.

5.5 Signal Dynamics and State Space

Our approach is general and works for any set of predictors \mathbf{z}_t that might be relevant at the time scale of portfolio rebalancing periods Δt . For example, for daily portfolio trading with time steps $\Delta t \simeq 1/250$, predictors \mathbf{z}_t may include news and various market indices such as VIX and MSCI indices. For portfolio trading on monthly or quarterly steps, additional predictors can include macroeconomic variables. In the opposite limit of intra-day or high-frequency trading, instead of macroeconomic variables, variables derived from the current state of the limit order book (LOB) might be more useful.

As a general rule, a predictor z_t may be of interest if it satisfies three requirements: (i) it correlates with equity returns, (ii) is predictable itself, to a certain degree (e.g., it can be a mean-reverting process); and (iii) its characteristic times τ are larger than the time step Δt . In particular, for a mean-reverting signal z_t , a mean reversion parameter κ gives rise to a characteristic time scale $\tau \simeq 1/\kappa$.

The last requirement simply means that if $\tau \ll \Delta t$ and the mean level of z_t is zero, then fluctuations of z_t will be well described by a stationary white noise process, and thus will be indistinguishable from the white noise term that is already present in Eq. (10.80). It is for this reason that it would be futile to, e.g., include

any features derived from the LOB for portfolio construction designed for monthly rebalancing.

For dynamics of signals \mathbf{z}_t , similar to Garleanu and Pedersen (2013), we will assume a simple multivariate mean-reverting Ornstein–Uhlenbeck (OU) process for a K -component vector \mathbf{z}_t :

$$\mathbf{z}_{t+1} = (\mathbf{I} - \Phi) \circ \mathbf{z}_t + \varepsilon_t^z, \quad (10.82)$$

where $\varepsilon_t^z \sim \mathcal{N}(0, \Sigma_z)$ is the noise term, and Φ is a diagonal matrix of mean reversion rates.

It is convenient to form an extended state vector \mathbf{y}_t of size $N + K$ by concatenating vectors \mathbf{x}_t and \mathbf{z}_t :

$$\mathbf{y}_t = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{z}_t \end{bmatrix}. \quad (10.83)$$

The extended vector, \mathbf{y}_t , describes a full state of the system for the agent that has some control of its x -component, but no control of its z -component.

5.6 One-Period Rewards

We first consider an idealized case when there are no costs of taking action \mathbf{u}_t at time step t . An instantaneous random reward received upon taking such action is obtained by substituting Eq. (10.80) in Eq. (10.78):

$$R_t^{(0)}(\mathbf{y}_t, \mathbf{u}_t) = \left(\mathbf{W}\mathbf{z}_t - \mathbf{M}^T \mathbf{u}_t + \boldsymbol{\varepsilon}_t \right)^T (\mathbf{x}_t + \mathbf{u}_t). \quad (10.84)$$

In addition to this reward that would be obtained in an ideal friction-free market, we must add (negative) rewards received due to instantaneous market impact and transaction fees.⁶ Furthermore, we must include a negative reward due to risk in a newly created portfolio position at time $t + 1$. Following Boyd et al. (2017), we choose a simple quadratic measure of such risk penalty, as the variance of the instantaneous reward (10.84) conditional on the new state $\mathbf{x}_t + \mathbf{u}_t$, multiplied by the risk-aversion parameter λ :

$$R_t^{(risk)}(\mathbf{y}_t, \mathbf{u}_t) = -\lambda \mathbb{V} \left[R_t^{(0)}(\mathbf{y}_t, \mathbf{u}_t) \mid \mathbf{x}_t + \mathbf{u}_t \right] = -\lambda (\mathbf{x}_t + \mathbf{u}_t)^T \Sigma_r (\mathbf{x}_t + \mathbf{u}_t). \quad (10.85)$$

To specify negative rewards (costs) of an instantaneous market impact and transaction costs, it is convenient to represent each action u_{ti} as a difference of two non-negative action variables $u_{ti}^+, u_{ti}^- \geq 0$:

⁶We assume no short sale positions in our setting, and therefore do not include borrowing costs.

$$u_{ti} = u_{ti}^+ - u_{ti}^-, \quad |u_{ti}| = u_{ti}^+ + u_{ti}^-, \quad u_{ti}^+, u_{ti}^- \geq 0, \quad (10.86)$$

so that $u_{ti} = u_{ti}^+$ if $u_{ti} > 0$ and $u_{ti} = -u_{ti}^-$ if $u_{ti} < 0$. The instantaneous market impact and transaction costs are then given by the following expressions:

$$\begin{aligned} R_t^{(impact)}(\mathbf{y}_t, \mathbf{u}_t) &= -\mathbf{x}_t^T \Gamma^+ \mathbf{u}_t^+ - \mathbf{x}_t^T \Gamma^- \mathbf{u}_t^- - \mathbf{x}_t^T \Upsilon \mathbf{z}_t \\ R_t^{(fee)}(\mathbf{y}_t, \mathbf{u}_t) &= -v^{+T} \mathbf{u}_t^+ - v^{-T} \mathbf{u}_t^-. \end{aligned} \quad (10.87)$$

Here Γ^+ , Γ^- , Υ and v^+ , v^- are, respectively, matrix-valued and vector-valued parameters which in the simplest case can be parameterized in terms of single scalars multiplied by unit vectors or matrices.

Combining Eqs. (10.84), (10.85), (10.87), we obtain our final specification of a risk- and cost-adjusted instantaneous reward function for the problem of optimal portfolio liquidation:

$$R_t(\mathbf{y}_t, \mathbf{u}_t) = R_t^{(0)}(\mathbf{y}_t, \mathbf{u}_t) + R_t^{(risk)}(\mathbf{y}_t, \mathbf{u}_t) + R_t^{(impact)}(\mathbf{y}_t, \mathbf{u}_t) + R_t^{(fee)}(\mathbf{y}_t, \mathbf{u}_t). \quad (10.88)$$

The *expected* one-step reward given action $\mathbf{u}_t = \mathbf{u}_t^+ - \mathbf{u}_t^-$ is given by

$$\hat{R}_t(\mathbf{y}_t, \mathbf{u}_t) = \hat{R}_t^{(0)}(\mathbf{y}_t, \mathbf{u}_t) + R_t^{(risk)}(\mathbf{y}_t, \mathbf{u}_t) + R_t^{(impact)}(\mathbf{y}_t, \mathbf{u}_t) + R_t^{(fee)}(\mathbf{y}_t, \mathbf{u}_t), \quad (10.89)$$

where

$$\hat{R}_t^{(0)}(\mathbf{y}_t, \mathbf{u}_t) = \mathbb{E}_{t,u} \left[R_t^{(0)}(\mathbf{y}_t, \mathbf{u}_t) \right] = \left(\mathbf{W} \mathbf{z}_t - \mathbf{M}^T (\mathbf{u}_t^+ - \mathbf{u}_t^-) \right)^T (\mathbf{x}_t + \mathbf{u}_t^+ - \mathbf{u}_t^-), \quad (10.90)$$

and where $\mathbb{E}_{t,u} [\cdot] := \mathbb{E} [\cdot | \mathbf{y}_t, \mathbf{u}_t]$ denotes averaging over next-periods realizations of market returns.

Note that the one-step expected reward (10.89) is a quadratic form of its inputs. We can write it more explicitly using vector notation:

$$\hat{R}(\mathbf{y}_t, \mathbf{a}_t) = \mathbf{y}_t^T \mathbf{R}_{yy} \mathbf{y}_t + \mathbf{a}_t^T \mathbf{R}_{aa} \mathbf{a} + \mathbf{a}_t^T \mathbf{R}_{ay} \mathbf{y}_t + \mathbf{a}_t^T \mathbf{R}_a, \quad (10.91)$$

where

$$\begin{aligned} \mathbf{a}_t &= \begin{pmatrix} \mathbf{u}_t^+ \\ \mathbf{u}_t^- \end{pmatrix}, \quad \mathbf{R}_{aa} = \begin{bmatrix} -\mathbf{M} - \lambda \Sigma_r & \mathbf{M} + \lambda \Sigma_r \\ \mathbf{M} + \lambda \Sigma_r & -\mathbf{M} - \lambda \Sigma_r \end{bmatrix}, \quad \mathbf{R}_{yy} = \begin{bmatrix} -\lambda \Sigma_r & \mathbf{W} - \Upsilon \\ 0 & 0 \end{bmatrix}, \\ \mathbf{R}_{ay} &= \left[\begin{bmatrix} -\mathbf{M} - 2\lambda \Sigma_r - \Gamma^+ \\ \mathbf{M} + 2\lambda \Sigma_r - \Gamma^- \end{bmatrix}, \begin{bmatrix} \mathbf{W} \\ \mathbf{W} \end{bmatrix} \right], \quad \mathbf{R}_a = - \begin{bmatrix} v^+ \\ v^+ \end{bmatrix}. \end{aligned} \quad (10.92)$$

5.7 Multi-period Portfolio Optimization

Multi-period portfolio optimization is equivalently formulated either as maximization of risk- and cost-adjusted returns, as in the Markowitz portfolio model, or as minimization of risk- and cost-adjusted trading costs. The latter specification is usually used in problems of optimal portfolio liquidation.

The multi-period risk- and cost-adjusted reward maximization problem is defined as

$$\text{maximize } \mathbb{E}_t \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} \hat{R}_{t'}(\mathbf{y}_{t'}, \mathbf{a}_{t'}) \right] \quad (10.93)$$

$$\text{where } \hat{R}_t(\mathbf{y}_t, \mathbf{a}_t) = \mathbf{y}_t^T \mathbf{R}_{yy} \mathbf{y}_t + \mathbf{a}_t^T \mathbf{R}_{aa} \mathbf{a}_t + \mathbf{a}_t^T \mathbf{R}_{ay} \mathbf{y}_t + \mathbf{a}_t^T \mathbf{R}_a$$

$$\text{w.r.t. } \mathbf{a}_t = \begin{pmatrix} \mathbf{u}_t^+ \\ \mathbf{u}_t^- \end{pmatrix} \geq 0,$$

$$\text{subject to } \mathbf{x}_t + \mathbf{u}_t^+ - \mathbf{u}_t^- \geq 0.$$

Here $0 < \gamma \leq 1$ is a discount factor. Note that the sum over future periods $t' = [t, \dots, T-1]$ does not include the last period $t' = T$, because the last action is fixed by Eq. (10.79).

The last constraint in Eq. (10.93) is appropriate for a long-only portfolio and can be replaced by other constraints, for example, a constraint on the portfolio leverage. With any (or both) of these constraints, the problem belongs to the class of convex optimization with constraints, and thus can be solved in a numerically efficient way (Boyd et al. 2017).

An equivalent cost-focused formulation is obtained by flipping the sign of the above problem and re-phrasing it as minimization of trading costs $\hat{C}_t(\mathbf{y}_t, \mathbf{a}_t) = -\hat{R}_t(\mathbf{y}_t, \mathbf{a}_t)$:

$$\text{minimize } \mathbb{E}_t \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} \hat{C}_{t'}(\mathbf{y}_{t'}, \mathbf{a}_{t'}) \right] \quad (10.94)$$

$$\text{where } \hat{C}_t(\mathbf{y}_t, \mathbf{a}_t) = -\hat{R}_t(\mathbf{y}_t, \mathbf{a}_t), \quad (10.95)$$

subject to the same constraints as in (10.93).

5.8 Stochastic Policy

Note that the multi-period portfolio optimization problem (10.93) assumes that an optimal policy that determines actions \mathbf{a}_t is a deterministic policy that can also be described as a delta-like probability distribution

$$\pi(\mathbf{a}_t | \mathbf{y}_t) = \delta(\mathbf{a}_t - \mathbf{a}_t^*(\mathbf{y}_t)), \quad (10.96)$$

where the optimal deterministic action $\mathbf{a}_t^*(\mathbf{y}_t)$ is obtained by maximization of the objective (10.93) with respect to controls \mathbf{a}_t .

But the actual trading data may be sub-optimal, or noisy at times, because of model mis-specifications, market timing lags, human errors, etc. Potential presence of such sub-optimal actions in data poses serious challenges, if we try to assume deterministic policy (10.96) that assumes the action chosen is *always* an optimal action. This is because such events should have zero probability under these model assumptions, and thus would produce vanishing path probabilities if observed in data.

Instead of assuming a deterministic policy (10.96), *stochastic* policies described by *smoothed* distributions $\pi(\mathbf{a}_t|\mathbf{y}_t)$ are more useful for inverse problems such as the problem of inverse portfolio optimization. In this approach, instead of maximization with respect to deterministic policy/action \mathbf{a}_t , we re-formulate the problem as maximization over *probability distributions* $\pi(\mathbf{a}_t|\mathbf{y}_t)$:

$$\text{maximize } \mathbb{E}_{q_\pi} \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} \hat{R}_t(\mathbf{y}_{t'}, \mathbf{a}_{t'}) \right] \quad (10.97)$$

$$\text{where } \hat{R}(\mathbf{y}_t, \mathbf{a}_t) = \mathbf{y}_t^T \mathbf{R}_{yy} \mathbf{y}_t + \mathbf{a}_t^T \mathbf{R}_{aa} \mathbf{a}_t + \mathbf{a}_t^T \mathbf{R}_{ay} \mathbf{y}_t + \mathbf{a}_t^T \mathbf{R}_a$$

$$\text{w.r.t. } q_\pi(\bar{x}, \bar{a}|\mathbf{y}_0) = \pi(\mathbf{a}_0|\mathbf{y}_0) \prod_{t=1}^{T-1} \pi(\mathbf{a}_t|\mathbf{y}_t) P(\mathbf{y}_{t+1}|\mathbf{y}_t, \mathbf{a}_t)$$

$$\text{subject to } \int d\mathbf{a}_t \pi(\mathbf{a}_t|\mathbf{y}_t) = 1.$$

Here $\mathbb{E}_{q_\pi} [\cdot]$ denotes expectations with respect to path probabilities defined according to the third line in Eqs. (10.97).

Note that due to inclusion of a quadratic risk penalty in the risk-adjusted return, $\hat{R}(\mathbf{x}_t, \mathbf{a}_t)$, the original problem of risk-adjusted return optimization is re-stated in Eq. (10.97) as maximizing the expected cumulative reward in the standard MDP setting, thus making the problem amenable to a standard risk-neutral approach of MDP models. Such simple risk adjustment based on one-step variance penalties was suggested in a non-financial context by Gosavi (2015) and used in a reinforcement learning based approach to option pricing in Halperin (2018, 2019).

Another comment that is due here is that a probabilistic approach to actions in portfolio trading appears, on many counts, a more natural approach than a formalism based on deterministic policies. Indeed, even in a simplest one-period setting, because the Markowitz-optimal solution for portfolio weights is a function of *estimated* stock means and covariances, they are in fact *random* variables. Yet the probabilistic nature of portfolio optimization is not recognized as such in the Markowitz-type single-period or multi-period optimization settings such as (10.93). A probabilistic portfolio optimization formulation was suggested in a one-period setting by Marschinski et al. (2007).

5.9 Reference Policy

We assume that we are given a probabilistic *reference* (or *prior*) policy $\pi_0(\mathbf{a}_t | \mathbf{y}_t)$ which should be decided upon prior to attempting the portfolio optimization (10.97). Such policy can be chosen based on a parametric model, past historic data, etc. We will use a simple Gaussian reference policy

$$\pi_0(\mathbf{a}_t | \mathbf{y}_t) = \frac{1}{\sqrt{(2\pi)^N |\Sigma_p|}} \exp\left(-\frac{1}{2} (\mathbf{a}_t - \hat{\mathbf{a}}(\mathbf{y}_t))^T \Sigma_p^{-1} (\mathbf{a}_t - \hat{\mathbf{a}}(\mathbf{y}_t))\right), \quad (10.98)$$

where $\hat{\mathbf{a}}(\mathbf{y}_t)$ can be a deterministic policy chosen to be a linear function of a state vector \mathbf{y}_t :

$$\hat{\mathbf{a}}(\mathbf{y}_t) = \hat{\mathbf{A}}_0 + \hat{\mathbf{A}}_1 \mathbf{y}_t. \quad (10.99)$$

A simple choice of parameters in (10.98) could be to specify them in terms of only two scalars \hat{a}_0 , \hat{a}_1 as follows: $\hat{\mathbf{A}}_0 = \hat{a}_0 \mathbb{1}_{|A|}$ and $\hat{\mathbf{A}}_1 = \hat{a}_1 \mathbb{1}_{|A| \times |A|}$, where $|A|$ is the size of vector \mathbf{a}_t , $\mathbb{1}_A$ and $\mathbb{1}_{A \times A}$ are, respectively, a vector and matrix made of ones. The scalars \hat{a}_0 and \hat{a}_1 would then serve as hyperparameters in our setting. Similarly, covariance matrix Σ_p for the prior policy can be taken to be a simple matrix with constant correlations ρ_p and constant variances σ_p .

As will be shown below, an *optimal* policy has the same Gaussian form as the prior policy (10.98), with updated parameters $\hat{\mathbf{A}}_0$, $\hat{\mathbf{A}}_1$, and Σ_p . These updates will be computed iteratively starting with their initial values defining the prior (10.98). Respectively, updates at iteration k will be denoted by upper subscripts, e.g. $\hat{\mathbf{A}}_0^{(k)}$, $\hat{\mathbf{A}}_1^{(k)}$.

Furthermore, it turns out that a linear dependence on \mathbf{y}_t at iteration k , driven by the value of $\hat{\mathbf{A}}_1^{(k)}$ arises even if we set $\hat{\mathbf{A}}_1 = \hat{\mathbf{A}}_1^{(0)} = 0$ in the prior (10.98). Such choice of a state-independent prior $\pi_0(\mathbf{a}_t | \mathbf{y}_t) = \pi_0(\mathbf{a}_t)$, although not very critical, reduces the number of free parameters in the model by two, as well as simplifies some of the analyses below, and hence will be assumed going forward. It also makes it unnecessary to specify the value of $\bar{\mathbf{y}}_t$ in the prior (10.98) (equivalently, we can initialize it at zero). The final set of hyperparameters defining the prior (10.98) therefore includes only three values of \hat{a}_0 , ρ_a , Σ_p .

5.10 Bellman Optimality Equation

Let

$$V_t^\star(\mathbf{y}_t) = \max_{\pi(\cdot | \mathbf{y})} \mathbb{E} \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} \hat{R}_{t'}(\mathbf{y}_{t'}, \mathbf{a}_{t'}) \middle| \mathbf{y}_t \right]. \quad (10.100)$$

The optimal state-value function $V_t^*(\mathbf{y}_t)$ satisfies the Bellman optimality equation

$$V_t^*(\mathbf{y}_t) = \max_{\mathbf{a}_t} \hat{R}_t(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}_t} [V_{t+1}^*(\mathbf{y}_{t+1})]. \quad (10.101)$$

The optimal policy π^* can be obtained from V^* as follows:

$$\pi_t^*(\mathbf{a}_t | \mathbf{y}_t) = \arg \max_{\mathbf{a}_t} \hat{R}_t(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}_t} [V_{t+1}^*(\mathbf{y}_{t+1})]. \quad (10.102)$$

The goal of reinforcement learning (RL) is to solve the Bellman optimality equation based on samples of data. Assuming that an optimal value function is found by means of RL, solving for the optimal policy π^* takes another optimization problem as formulated in Eq. (10.102).

5.11 Entropy-Regularized Bellman Optimality Equation

We start by reformulating the Bellman optimality equation using a Fenchel-type representation:

$$V_t^*(\mathbf{y}_t) = \max_{\pi(\cdot | \mathbf{y}_t) \in \mathcal{P}} \sum_{\mathbf{a}_t \in \mathcal{A}_t} \pi(\mathbf{a}_t | \mathbf{y}_t) \left(\hat{R}_t(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}_t} [V_{t+1}^*(\mathbf{y}_{t+1})] \right). \quad (10.103)$$

Here $\mathcal{P} = \{\pi : \pi \geq 0, \mathbb{1}^T \pi = 1\}$ denotes a set of all valid distributions. Equation (10.103) is equivalent to the original Bellman optimality equation (10.100), because for any $x \in \mathbb{R}^n$, we have $\max_{i \in \{1, \dots, n\}} x_i = \max_{\pi \geq 0, \|\pi\| \leq 1} \pi^T x$. Note that while we use discrete notations for simplicity of presentation, all formulas below can be equivalently expressed in continuous notations by replacing sums by integrals. For brevity, we will denote the expectation $\mathbb{E}_{\mathbf{y}_{t+1} | \mathbf{y}_t, \mathbf{a}_t} [\cdot]$ as $\mathbb{E}_{t, \mathbf{a}} [\cdot]$ in what follows.

The one-step *information cost* of a learned policy $\pi(\mathbf{a}_t | \mathbf{y}_t)$ relative to a reference policy $\pi_0(\mathbf{a}_t | \mathbf{y}_t)$ is defined as follows (Fox et al. 2015):

$$g^\pi(\mathbf{y}, \mathbf{a}) = \log \frac{\pi(\mathbf{a}_t | \mathbf{y}_t)}{\pi_0(\mathbf{a}_t | \mathbf{y}_t)}. \quad (10.104)$$

Its expectation with respect to the policy π is the Kullback–Leibler (KL) divergence of $\pi(\cdot | \mathbf{y}_t)$ and $\pi_0(\cdot | \mathbf{y}_t)$:

$$\mathbb{E}_\pi [g^\pi(\mathbf{y}, \mathbf{a}) | \mathbf{y}_t] = KL[\pi || \pi_0](\mathbf{y}_t) := \sum_{\mathbf{a}_t} \pi(\mathbf{a}_t | \mathbf{y}_t) \log \frac{\pi(\mathbf{a}_t | \mathbf{y}_t)}{\pi_0(\mathbf{a}_t | \mathbf{y}_t)}. \quad (10.105)$$

The total discounted information cost for a trajectory is defined as follows:

$$I^\pi(\mathbf{y}) = \sum_{t'=t}^T \gamma^{t'-t} \mathbb{E} [g^\pi(\mathbf{y}_{t'}, \mathbf{a}_{t'}) \mid \mathbf{y}_t = \mathbf{y}]. \quad (10.106)$$

The *free energy* function $F_t^\pi(\mathbf{y}_t)$ is defined as the value function (10.103) augmented by the information cost penalty (10.106):

$$\begin{aligned} F_t^\pi(\mathbf{y}_t) &= V_t^\pi(\mathbf{y}_t) - \frac{1}{\beta} I^\pi(\mathbf{y}_t) \\ &= \sum_{t'=t}^T \gamma^{t'-t} \mathbb{E} \left[\hat{R}_{t'}(\mathbf{y}_{t'}, \mathbf{a}_{t'}) - \frac{1}{\beta} g^\pi(\mathbf{y}_{t'}, \mathbf{a}_{t'}) \right]. \end{aligned} \quad (10.107)$$

Note that β in Eq. (10.107) serves as the “inverse temperature” parameter that controls a tradeoff between reward optimization and proximity to the reference policy, see below. The free energy, $F_t^\pi(\mathbf{y}_t)$, is the entropy-regularized value function, where the amount of regularization can be tuned to the level of noise in the data.⁷ The reference policy, π_0 , provides a “guiding hand” in the stochastic policy optimization process that we now describe.

A Bellman equation for the free energy function $F_t^\pi(\mathbf{y}_t)$ is obtained from (10.107):

$$F_t^\pi(\mathbf{y}_t) = \mathbb{E}_{\mathbf{a}|\mathbf{y}} \left[\hat{R}_t(\mathbf{y}_t, \mathbf{a}_t) - \frac{1}{\beta} g^\pi(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t+1, \mathbf{a}} [F_{t+1}^\pi(\mathbf{y}_{t+1})] \right]. \quad (10.108)$$

For a finite-horizon setting, Eq. (10.108) should be supplemented by a terminal condition

$$F_T^\pi(\mathbf{y}_T) = \hat{R}_T(\mathbf{y}_T, \mathbf{a}_T) \Big|_{\mathbf{a}_T = -\mathbf{x}_{T-1}} \quad (10.109)$$

(see Eq. (10.79)). Eq. (10.108) can be viewed as a soft probabilistic relaxation of the Bellman optimality equation for the value function, with the KL information cost penalty (10.104) as a regularization controlled by the inverse temperature β . In addition to such a regularized value function (free energy), we will next introduce an entropy regularized Q-function.

⁷Note that in physics, free energy is defined with a negative sign relative to Eq. (10.107). This difference is purely a matter of a sign convention, as maximization of Eq. (10.107) can be re-stated as minimization of its negative. Using our sign convention for the free energy function, we follow the reinforcement learning and information theory literature.

5.12 G-Function: An Entropy-Regularized Q-Function

Similar to the action-value function, we define the state-action free energy function $G^\pi(\mathbf{y}, \mathbf{a})$ as (Fox et al. 2015)

$$\begin{aligned} G_t^\pi(\mathbf{y}_t, \mathbf{a}_t) &= \hat{R}_t(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E} [F_{t+1}^\pi(\mathbf{y}_{t+1}) | \mathbf{y}_t, \mathbf{a}_t] \\ &= \hat{R}_t(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}} \left[\sum_{t'=t+1}^T \gamma^{t'-t-1} \left(\hat{R}_{t'}(\mathbf{y}_{t'}, \mathbf{a}_{t'}) - \frac{1}{\beta} g^\pi(\mathbf{y}_{t'}, \mathbf{a}_{t'}) \right) \right] \\ &= \mathbb{E}_{t, \mathbf{a}} \left[\sum_{t'=t}^T \gamma^{t'-t} \left(\hat{R}_{t'}(\mathbf{y}_{t'}, \mathbf{a}_{t'}) - \frac{1}{\beta} g^\pi(\mathbf{y}_{t'}, \mathbf{a}_{t'}) \right) \right], \end{aligned} \quad (10.110)$$

where in the last equation we used the fact that the first action \mathbf{a}_t in the G-function is fixed, and hence $g^\pi(\mathbf{y}_t, \mathbf{a}_t) = 0$ when we condition on $\mathbf{a}_t = \mathbf{a}$.

If we now compare this expression with Eq. (10.107), we obtain the relation between the G-function and the free energy $F_t^\pi(\mathbf{y}_t)$:

$$F_t^\pi(\mathbf{y}_t) = \sum_{\mathbf{a}_t} \pi(\mathbf{a}_t | \mathbf{y}_t) \left[G_t^\pi(\mathbf{y}_t, \mathbf{a}_t) - \frac{1}{\beta} \log \frac{\pi(\mathbf{a}_t | \mathbf{y}_t)}{\pi_0(\mathbf{a}_t | \mathbf{y}_t)} \right]. \quad (10.111)$$

This functional is maximized by the following distribution $\pi(\mathbf{a}_t | \mathbf{y}_t)$:

$$\begin{aligned} \pi(\mathbf{a}_t | \mathbf{y}_t) &= \frac{1}{Z_t} \pi_0(\mathbf{a}_t | \mathbf{y}_t) e^{\beta G_t^\pi(\mathbf{y}_t, \mathbf{a}_t)} \\ Z_t &= \sum_{\mathbf{a}_t} \pi_0(\mathbf{a}_t | \mathbf{y}_t) e^{\beta G_t^\pi(\mathbf{y}_t, \mathbf{a}_t)}. \end{aligned} \quad (10.112)$$

The free energy (10.111) evaluated at the optimal solution (10.112) becomes

$$F_t^\pi(\mathbf{y}_t) = \frac{1}{\beta} \log Z_t = \frac{1}{\beta} \log \sum_{\mathbf{a}_t} \pi_0(\mathbf{a}_t | \mathbf{y}_t) e^{\beta G_t^\pi(\mathbf{y}_t, \mathbf{a}_t)}. \quad (10.113)$$

Using Eq. (10.113), the optimal action policy can be written as follows :

$$\pi(\mathbf{a}_t | \mathbf{y}_t) = \pi_0(\mathbf{a}_t | \mathbf{y}_t) e^{\beta(G_t^\pi(\mathbf{y}_t, \mathbf{a}_t) - F_t^\pi(\mathbf{y}_t))}. \quad (10.114)$$

Equations (10.113), (10.114), along with the first form of Eq. (10.110) repeated here for convenience:

$$G_t^\pi(\mathbf{y}_t, \mathbf{a}_t) = \hat{R}_t(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}} [F_{t+1}^\pi(\mathbf{y}_{t+1}) | \mathbf{y}_t, \mathbf{a}_t], \quad (10.115)$$

constitute a system of equations that should be solved self-consistently by backward recursion for $t = T - 1, \dots, 0$, with terminal conditions

$$\begin{aligned} G_T^\pi(\mathbf{y}_T, \mathbf{a}_T) &= \hat{R}_T(\mathbf{y}_T, \mathbf{a}_T) \\ F_T^\pi(\mathbf{y}_T) &= G_T^\pi(\mathbf{y}_T, \mathbf{a}_T) = \hat{R}_T(\mathbf{y}_T, \mathbf{a}_T). \end{aligned} \quad (10.116)$$

Equations (10.113, 10.114, 10.115) (Fox et al. 2015) constitute a system of equations that should be solved self-consistently for $\pi(\mathbf{a}_t|\mathbf{y}_t)$, $G_t^\pi(\mathbf{y}_t, \mathbf{a}_t)$, and $F_t^\pi(\mathbf{y}_t)$. Before proceeding with methods of solving it, we want to digress on an alternative interpretation of entropy regularization in Eq. (10.107), that can be useful later in the book.

> Adversarial Interpretation of Entropy Regularization

A useful alternative interpretation of the entropy regularization term in Eq. (10.107) can be suggested using its representation as a Legendre–Fenchel transform of another function (Ortega and Lee 2014):

$$\begin{aligned} -\frac{1}{\beta} \sum_{\mathbf{a}_t} \pi(\mathbf{a}_t|\mathbf{y}_t) \log \frac{\pi(\mathbf{a}_t|\mathbf{y}_t)}{\pi_0(\mathbf{a}_t|\mathbf{y}_t)} &= \min_{C(\mathbf{a}_t, \mathbf{y}_t)} \sum_{\mathbf{a}_t} (-\pi(\mathbf{a}_t|\mathbf{y}_t) (1 + C(\mathbf{a}_t, \mathbf{y}_t)) \\ &\quad + \pi_0(\mathbf{a}_t|\mathbf{y}_t) e^{\beta C(\mathbf{a}_t, \mathbf{y}_t)}), \end{aligned} \quad (10.117)$$

where $C(\mathbf{a}_t, \mathbf{y}_t)$ is an arbitrary function. Equation (10.117) can be verified by direct minimization of the right-hand side with respect to $C(\mathbf{a}_t, \mathbf{y}_t)$.

Using this representation of the KL term, the free energy maximization problem (10.111) can be re-stated as a max–min problem

$$F_t^*(\mathbf{y}_t) = \max_{\pi} \min_C \sum_{\mathbf{a}_t} \pi(\mathbf{a}_t|\mathbf{y}_t) [G_t^\pi(\mathbf{y}_t, \mathbf{a}_t) - C(\mathbf{a}_t, \mathbf{y}_t) - 1] + \pi_0(\mathbf{a}_t|\mathbf{y}_t) e^{\beta C(\mathbf{a}_t, \mathbf{y}_t)}. \quad (10.118)$$

The imaginary adversary's optimal cost obtained from (10.118) is

$$C^*(\mathbf{a}_t, \mathbf{y}_t) = \frac{1}{\beta} \log \frac{\pi(\mathbf{a}_t|\mathbf{y}_t)}{\pi_0(\mathbf{a}_t|\mathbf{y}_t)}. \quad (10.119)$$

Similar to Ortega and Lee (2014), one can check that this produces an *indifference* solution for the imaginary game between the agent and its adversarial environment where the total sum of the optimal G-function and the

(continued)

optimal adversarial cost (10.119) is constant: $G_t^*(\mathbf{y}_t, \mathbf{a}_t) + C^*(\mathbf{a}_t, \mathbf{y}_t) = \text{const}$, which means that the game of the original agent and its adversary is in a Nash equilibrium .

Therefore, portfolio optimization in a stochastic environment by a single agent is mathematically equivalent to studying a Nash equilibrium in a two-party game of our agent with an adversarial counterpart with an exponential budget given by the last term in Eq. (10.118).

5.13 G-Learning and F-Learning

In the RL setting when rewards are observed, the system Eqs. (10.113, 10.114, 10.115) can be reduced to one non-linear equation. Substituting the augmented free energy (10.113) into Eq. (10.115), we obtain

$$G_t^\pi(\mathbf{y}, \mathbf{a}) = \hat{R}(\mathbf{y}_t, \mathbf{a}_t) + \mathbb{E}_{t,\mathbf{a}} \left[\frac{\gamma}{\beta} \log \sum_{\mathbf{a}_{t+1}} \pi_0(\mathbf{a}_{t+1} | \mathbf{y}_{t+1}) e^{\beta G_{t+1}^\pi(\mathbf{y}_{t+1}, \mathbf{a}_{t+1})} \right]. \quad (10.120)$$

This equation provides a soft relaxation of the Bellman optimality equation for the action-value Q-function, with the G-function defined in Eq. (10.110) being an entropy-regularized Q-function (Fox et al. 2015). The “inverse-temperature” parameter β in Eq. (10.120) determines the strength of entropy regularization. In particular, if we take $\beta \rightarrow \infty$, we recover the original Bellman optimality equation for the Q-function. Because the last term in (10.120) approximates the $\max(\cdot)$ function when β is large but finite, Eq. (10.120) is known, for the special case of a uniform reference policy π_0 , as “soft Q-learning”.

For finite values $\beta < \infty$, in a setting of reinforcement learning with observed rewards, Eq. (10.120) can be used to specify *G-learning* (Fox et al. 2015): an off-policy time-difference (TD) algorithm that generalizes Q-learning to noisy environments where an entropy-based regularization might be needed. The G-learning algorithm of Fox et al. (2015) was specified in a tabulated setting where both the state and action space are finite. In our case, we deal with high-dimensional continuous state and action spaces. Respectively, we cannot rely on a tabulated G-learning and need to specify a functional form of the action-value function, or use a non-parametric function approximation such as a neural network to represent its values. An additional challenge is to compute a multi-dimensional integral (or a sum) over all next-step actions in Eq. (10.120). Unless a tractable parameterization is used for π_0 and G_t , repeated numerical integration of this integral can substantially slow down the learning.

➤ G-Learning vs Q-Learning

- *Q-learning* is an off-policy method with a *deterministic* policy.
- *G-learning* is an off-policy method with a *stochastic* policy. Because the G-learning operates with stochastic policies, it gives rise to generative models. G-learning can be considered as an entropy-regularized Q-learning.

Another possible approach is to bypass the *G*-function (i.e., the entropy-regulated Q-function) altogether, and proceed with the Bellman optimality equation for the free energy F-function (10.107). In this case, we have a pair of equations for $F_t^\pi(\mathbf{y}_t)$ and $\pi(\mathbf{a}_t|\mathbf{y}_t)$:

$$\begin{aligned} F_t^\pi(\mathbf{y}_t) &= \mathbb{E}_{\mathbf{a}|x} \left[\hat{R}(\mathbf{y}_t, \mathbf{a}_t) - \frac{1}{\beta} g^\pi(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t,\mathbf{a}} [F_{t+1}^\pi(\mathbf{y}_{t+1})] \right] \\ \pi(\mathbf{a}_t|\mathbf{y}_t) &= \frac{1}{Z_t} \pi_0(\mathbf{a}_t|\mathbf{y}_t) e^{\hat{R}(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t,\mathbf{a}} [F_{t+1}^\pi(\mathbf{y}_{t+1})]}. \end{aligned} \quad (10.121)$$

Here the first equation is the Bellman equation (10.108) for the F-function, and the second equation is obtained by substitution of Eq. (10.115) into Eq. (10.112). Also note that the normalization constant Z_t in Eq. (10.121) is in general different from the normalization constant in Eq. (10.112).

We will return to solutions of G-learning with continuous states and actions in the next chapter where we will address inverse reinforcement learning.

➤ G-Learning for Stationary Problems

For time-stationary (infinite-horizon) problems, the “soft Q-learning” (G-learning) equation (10.120) becomes

(continued)

$$G^\pi(\mathbf{y}, \mathbf{a}) = \hat{R}(\mathbf{y}, \mathbf{a}) + \frac{\gamma}{\beta} \sum_{\mathbf{y}'} \rho(\mathbf{y}'|\mathbf{y}, \mathbf{a}) \left[\log \sum_{\mathbf{a}'} \pi_0(\mathbf{a}'|\mathbf{y}') e^{\beta G^\pi(\mathbf{y}', \mathbf{a}')} \right] \quad (10.122)$$

This is a non-linear integral equation. For example, if both the state and action space are one-dimensional, the resulting integral equation is two-dimensional. Therefore, computationally, G-learning for time-stationary problems amounts to solution of a non-linear integral equation (10.122). Existing numerical methods could be used to address this problem, see also Exercise 10.4.

5.14 Portfolio Dynamics with Market Impact

A state equation for the portfolio vector \mathbf{x}_t is obtained using Eqs.(10.74) and (10.80):

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{x}_t + \mathbf{u}_t + \mathbf{r}_t \circ (\mathbf{x}_t + \mathbf{u}_t) \\ &= \mathbf{x}_t + \mathbf{u}_t + (r_f \mathbb{1} + \mathbf{Wz}_t - \mathbf{M}^T \mathbf{u}_t + \varepsilon_t) \circ (\mathbf{x}_t + \mathbf{u}_t) \\ &= (1 + r_f)(\mathbf{x}_t + \mathbf{u}_t) + \text{diag}(\mathbf{Wz}_t - \mathbf{Mu}_t)(\mathbf{x}_t + \mathbf{u}_t) + \varepsilon(\mathbf{x}_t, \mathbf{u}_t) \end{aligned} \quad (10.123)$$

Here we assumed that the matrix M of market impacts is diagonal with elements μ_i , and set

$$\mathbf{M} = \text{diag}(\mu_i), \quad \varepsilon(\mathbf{x}_t, \mathbf{u}_t) := \varepsilon_t \circ (\mathbf{x}_t + \mathbf{u}_t) \quad (10.124)$$

Eq. (10.123) shows that the dynamics are non-linear in controls \mathbf{u}_t due to the market impact $\sim \mathbf{M}$. More specifically, when friction parameters $\mu_i > 0$, the state equation is linear in \mathbf{x}_t , but quadratic in controls \mathbf{u}_t . In the limit $\mu \rightarrow 0$, the dynamics become linear. On the other hand, the reward (10.91) is quadratic for either case $\mu_i = 0$ or $\mu_i > 0$.

The fact that the dynamics are non-linear (quadratic) when $\mu_i > 0$ has far-reaching implications both on the practical (computational) and theoretical side. Before discussing the non-linear case, we want to first analyze a simpler case when $\mu_i = 0$, i.e. market impact effects are neglected, and the dynamics are linear.

When dynamics are linear while rewards are quadratic, the problem of optimal portfolio management with a deterministic policy amounts to a well-known linear quadratic regulator (LQR) whose solution is known from control theory. In the next section we present a probabilistic version of the LQR problem that is particularly well suited for dynamic portfolio optimization.

5.15 Zero Friction Limit: LQR with Entropy Regularization

When the market impact is neglected, so that $\mu_i = 0$ for all i , the portfolio optimization problem simplifies because dynamics become linear with the following state equation:

$$\mathbf{x}_{t+1} = (1 + r_f + \mathbf{W}\mathbf{z}_t + \varepsilon_t) \circ (\mathbf{x}_t + \mathbf{u}_t). \quad (10.125)$$

We can equivalently write it as follows:

$$\mathbf{x}_{t+1} = \mathbf{A}_t(\mathbf{x}_t + \mathbf{u}_t) + (\mathbf{x}_t + \mathbf{u}_t) \circ \varepsilon_t \quad (10.126)$$

where

$$\mathbf{A}_t = \mathbf{A}(\mathbf{z}_t) = \text{diag}(1 + r_f + \mathbf{W}\mathbf{z}_t) \quad (10.127)$$

Unlike the previous section where we assumed proportional transaction costs, here we assume convex transaction costs $\eta \mathbf{u}_t^T \mathbf{C} \mathbf{u}_t$, where η is a transaction cost parameter and \mathbf{C} is a matrix which can be taken to be a diagonal unit matrix. We neglect other costs such as holding costs. The expected one-step reward at time t is then given by the following expression:

$$\hat{R}_t(\mathbf{x}_t, \mathbf{u}_t) = (\mathbf{x}_t + \mathbf{u}_t)^T \mathbf{W}\mathbf{z}_t - \lambda (\mathbf{x}_t + \mathbf{u}_t)^T \Sigma_r (\mathbf{x}_t + \mathbf{u}_t) - \eta \mathbf{u}_t^T \mathbf{C} \mathbf{u}_t. \quad (10.128)$$

If we assume that our problem is to maximize the risk-adjusted return of the portfolio for a pre-specified time horizon T , then the natural terminal condition for \mathbf{x}_T would be to set $\mathbf{x}_T = 0$, meaning that all stock positions should be converted to cash at maturity of the portfolio. This implies that the last action is deterministic rather than stochastic and is determined by the stock holding at time $T - 1$:

$$\mathbf{u}_{T-1} = \mathbf{x}_T - \mathbf{x}_{T-1} = -\mathbf{x}_{T-1}. \quad (10.129)$$

The last reward is therefore a quadratic functional of \mathbf{x}_{T-1} :

$$\hat{R}_{T-1} = -\eta \mathbf{u}_{T-1}^T \mathbf{C} \mathbf{u}_{T-1}. \quad (10.130)$$

As the last action is deterministic, optimization amounts to choosing the remaining $T - 1$ portfolio adjustments $\mathbf{u}_0, \dots, \mathbf{u}_{T-2}$.

We now show that reinforcement learning with G-learning can be solved semi-analytically in this setting using Gaussian time-varying policies (GTVP). We start by specifying a functional form of the value function as a quadratic form of \mathbf{x}_t :

$$F_t^\pi(\mathbf{x}_t) = \mathbf{x}_t^T \mathbf{F}_t^{(xx)} \mathbf{x}_t + \mathbf{x}_t^T \mathbf{F}_t^{(x)} + F_t^{(0)}, \quad (10.131)$$

where $\mathbf{F}_t^{(xx)}$, $\mathbf{F}_t^{(x)}$, $F_t^{(0)}$ are parameters that depend on time both explicitly (for finite horizon problems) and implicitly, via their dependence on the signals \mathbf{z}_t .

As for the last time step we have $F_{T-1}^\pi(\mathbf{x}_{T-1}) = \hat{R}_{T-1}$, using Eqs. (10.130) and (10.131), we obtain the terminal conditions for coefficients in Eq. (10.131):

$$\mathbf{F}_{T-1}^{(xx)} = -\eta \mathbf{C}, \quad \mathbf{F}_{T-1}^{(x)} = 0, \quad \mathbf{F}_{T-1}^{(0)} = 0. \quad (10.132)$$

For an arbitrary time step $t = T-2, \dots, 0$, we use Eq. (10.126) and independence of noise terms for \mathbf{x}_t and \mathbf{z}_t to compute the conditional expectation of the next-period F-function in Eq. (10.115) as follows:

$$\begin{aligned} \mathbb{E}_{t,\mathbf{a}}[F_{t+1}^\pi(\mathbf{x}_{t+1})] &= (\mathbf{x}_t + \mathbf{u}_t)^T \left(\mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(xx)} \mathbf{A}_t + \Sigma_r \circ \bar{\mathbf{F}}_{t+1}^{(xx)} \right) (\mathbf{x}_t + \mathbf{u}_t) \\ &\quad + (\mathbf{x}_t + \mathbf{u}_t)^T \mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(x)} + \bar{\mathbf{F}}_{t+1}^{(0)}, \end{aligned} \quad (10.133)$$

where $\bar{\mathbf{F}}_{t+1}^{(xx)} := \mathbb{E}_t[\mathbf{F}_{t+1}^{(xx)}]$, and similarly for $\bar{\mathbf{F}}_{t+1}^{(x)}$ and $\bar{\mathbf{F}}_{t+1}^{(0)}$. Importantly, this is a quadratic function of \mathbf{x}_t and \mathbf{u}_t . Combining it with the quadratic reward $\hat{R}(\mathbf{x}_t, \mathbf{a}_t)$ in (10.131) in the Bellman equation (10.115), we see that the action-value function $G_t^\pi(\mathbf{x}_t, \mathbf{u}_t)$ should also be a quadratic function of \mathbf{x}_t and \mathbf{u}_t :

$$G_t^\pi(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^T \mathbf{Q}_t^{(xx)} \mathbf{x}_t + \mathbf{u}_t^T \mathbf{Q}_t^{(uu)} \mathbf{u}_t + \mathbf{u}_t^T \mathbf{Q}_t^{(ux)} \mathbf{x}_t + \mathbf{u}_t^T \mathbf{Q}_t^{(u)} + \mathbf{x}_t^T \mathbf{Q}_t^{(x)} + Q_t^{(0)}, \quad (10.134)$$

where

$$\begin{aligned} \mathbf{Q}_t^{(xx)} &= -\lambda \Sigma_r + \gamma \left(\mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(xx)} \mathbf{A}_t + \Sigma_r \circ \bar{\mathbf{F}}_{t+1}^{(xx)} \right) \\ \mathbf{Q}_t^{(uu)} &= -\eta \mathbf{C} + \mathbf{Q}_t^{(xx)} \\ \mathbf{Q}_t^{(ux)} &= 2\mathbf{Q}_t^{(xx)} \\ \mathbf{Q}_t^{(x)} &= \mathbf{W}\mathbf{z}_t + \gamma \mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(x)} \\ \mathbf{Q}_t^{(u)} &= \mathbf{Q}_t^{(x)} \\ Q_t^{(0)} &= \gamma F_{t+1}^{(0)}. \end{aligned} \quad (10.135)$$

Having computed the G-function $G_t^\pi(\mathbf{x}_t, \mathbf{u}_t)$ in terms of its coefficients (10.135), the F-function for the current step can be found using Eq. (10.113) which we repeat here in terms of the original variables \mathbf{x}_t , \mathbf{u}_t , and replacing summation by integration:

$$F_t^\pi(\mathbf{x}_t) = \frac{1}{\beta} \log \int \pi_0(\mathbf{u}_t | \mathbf{x}_t) e^{\beta G_t^\pi(\mathbf{x}_t, \mathbf{u}_t)} d\mathbf{u}_t. \quad (10.136)$$

We assume that a reference policy $\pi_0(\mathbf{u}_t | \mathbf{x}_t)$ is Gaussian:

$$\pi_0(\mathbf{u}_t | \mathbf{x}_t) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_p|}} e^{-\frac{1}{2}(\mathbf{u}_t - \hat{\mathbf{u}}_t)^T \Sigma_p^{-1} (\mathbf{u}_t - \hat{\mathbf{u}}_t)}, \quad (10.137)$$

where the mean value $\hat{\mathbf{u}}_t$ is a linear function of the state \mathbf{x}_t :

$$\hat{\mathbf{u}}_t = \bar{\mathbf{u}}_t + \bar{\mathbf{v}}_t \mathbf{x}_t. \quad (10.138)$$

Here $\bar{\mathbf{u}}_t$ and $\bar{\mathbf{v}}_t$ are parameters that can be considered time-independent (so that the time index can be omitted) in the prior distribution (10.137). The reason we keep the time label is that, as we will see shortly, the optimal policy obtained from G-learning with linear dynamics (10.126) is also a Gaussian that can be written in the same form as (10.137) but with updated parameters $\bar{\mathbf{u}}_t$ and $\bar{\mathbf{v}}_t$ that will become time-dependent due to their dependence on signals \mathbf{z}_t .

If no constraints are imposed on \mathbf{u}_t , integration over \mathbf{u}_t in Eq. (10.136) with a Gaussian reference policy π_0 can be easily performed analytically as long as $G_t^\pi(\mathbf{x}_t, \mathbf{u}_t)$ is quadratic in \mathbf{u}_t .⁸ Using the n -dimensional Gaussian integration formula gives:

$$\int e^{-\frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{B}} d^n \mathbf{x} = \sqrt{\frac{(2\pi)^n}{|\mathbf{A}|}} e^{\frac{1}{2} \mathbf{B}^T \mathbf{A}^{-1} \mathbf{B}}, \quad (10.139)$$

where $|\mathbf{A}|$ denotes the determinant of matrix \mathbf{A} . Using this relation to calculate the integral in Eq. (10.136) and introducing auxiliary parameters

$$\begin{aligned} \mathbf{U}_t &= \beta \mathbf{Q}_t^{(ux)} + \Sigma_p^{-1} \bar{\mathbf{v}}_t \\ \mathbf{W}_t &= \beta \mathbf{Q}_t^{(u)} + \Sigma_p^{-1} \bar{\mathbf{u}}_t \\ \bar{\Sigma}_p &= \Sigma_p^{-1} - 2\beta \mathbf{Q}_t^{(uu)}, \end{aligned} \quad (10.140)$$

we find that the resulting F -function has the same structure as in Eq. (10.131), where the coefficients are now *computed* in terms of coefficients of the Q -function (see Exercise 10.3):

$$\begin{aligned} F_t^\pi(\mathbf{x}_t) &= \mathbf{x}_t^T \mathbf{F}_t^{(xx)} \mathbf{x}_t + \mathbf{x}_t^T \mathbf{F}_t^{(x)} + F_t^{(0)} \\ \mathbf{F}_t^{(xx)} &= \mathbf{Q}_t^{(xx)} + \frac{1}{2\beta} \left(\mathbf{U}_t^T \bar{\Sigma}_p^{-1} \mathbf{U}_t - \bar{\mathbf{v}}_t^T \Sigma_p^{-1} \bar{\mathbf{v}}_t \right) \end{aligned}$$

⁸As in the present formulation actions *are* constrained by the self-financing condition, an independent Gaussian integration may produce inaccurate results. For a constrained version of the integral with a constraint on the sum of variables, see Exercise 10.6. In the next section we will present a case where an unconstrained Gaussian integration works better.

$$\begin{aligned}\mathbf{F}_t^{(x)} &= \mathbf{Q}_t^{(x)} + \frac{1}{\beta} \left(\mathbf{U}_t^T \bar{\Sigma}_p^{-1} \mathbf{W}_t - \bar{\mathbf{v}}_t^T \Sigma_p^{-1} \bar{\mathbf{u}}_t \right) \\ \mathbf{F}_t^{(0)} &= \mathbf{Q}_t^{(0)} + \frac{1}{2\beta} \left(\mathbf{W}_t^T \bar{\Sigma}_p^{-1} \mathbf{W}_t - \bar{\mathbf{u}}_t^T \Sigma_p^{-1} \bar{\mathbf{u}}_t \right) - \frac{1}{2\beta} (\log |\Sigma_p| + \log |\bar{\Sigma}_p|).\end{aligned}\quad (10.141)$$

Finally, the optimal policy for the given step can be found using Eq. (10.114) which we again rewrite here in terms of the original variables \mathbf{x}_t , \mathbf{u}_t :

$$\pi(\mathbf{u}_t | \mathbf{x}_t) = \pi_0(\mathbf{u}_t | \mathbf{x}_t) e^{\beta(G_t^\pi(\mathbf{x}_t, \mathbf{u}_t) - F_t^\pi(\mathbf{x}_t))}. \quad (10.142)$$

As $G_t^\pi(\mathbf{x}_t, \mathbf{u}_t)$ is a quadratic function of \mathbf{u}_t , this produces a Gaussian policy $\pi(\mathbf{u}_t | \mathbf{x}_t)$:

$$\pi(\mathbf{u}_t | \mathbf{x}_t) = \frac{1}{\sqrt{(2\pi)^n |\tilde{\Sigma}_p|}} e^{-\frac{1}{2}(\mathbf{u}_t - \bar{\mathbf{u}}_t - \bar{\mathbf{v}}_t \mathbf{x}_t)^T \tilde{\Sigma}_p^{-1} (\mathbf{u}_t - \bar{\mathbf{u}}_t - \bar{\mathbf{v}}_t \mathbf{x}_t)}, \quad (10.143)$$

with updated parameters

$$\begin{aligned}\tilde{\Sigma}_p^{-1} &= \Sigma_p^{-1} - 2\beta \mathbf{Q}_t^{(uu)} \\ \bar{\mathbf{u}}_t &= \tilde{\Sigma}_p \left(\Sigma_p^{-1} \bar{\mathbf{u}}_t + \beta \mathbf{Q}_t^{(u)} \right) \\ \bar{\mathbf{v}}_t &= \tilde{\Sigma}_p \left(\Sigma_p^{-1} \bar{\mathbf{v}}_t + \beta \mathbf{Q}_t^{(ux)} \right).\end{aligned}\quad (10.144)$$

Therefore, policy optimization with the entropy-regularized LQR with signals expressed by Eq. (10.142) amounts to the Bayesian update of the prior distribution (10.137) with parameters updates $\bar{\mathbf{u}}_t$, $\bar{\mathbf{v}}_t$, Σ_p to the new values $\tilde{\mathbf{u}}_t$, $\tilde{\mathbf{v}}_t$, $\tilde{\Sigma}_p$ defined in Eqs. (10.144). These quantities depend on time via their dependence on \mathbf{z}_t .

Note that the third of equations (10.144) indicates that even if we started with $\bar{\mathbf{v}}_t = 0$ (meaning a state-independent mean level in Eq. (10.137)), the optimal policy has a mean that is linear in state \mathbf{x}_t as long as $\mathbf{Q}_t^{(ux)} \neq 0$. Therefore, the entropy-regularized LQR produces a Gaussian optimal policy whose mean is a linear function of the state \mathbf{x}_t . This provides a probabilistic generalization of a regular (deterministic) LQR where an optimal policy itself is a linear function of the state.

Another interesting point to note about the Bayesian update (10.144) is that even if we start with time-independent values $\bar{\mathbf{u}}_t$, $\bar{\mathbf{v}}_t = \bar{\mathbf{u}}$, $\bar{\mathbf{v}}$, the updated values $\tilde{\mathbf{u}}_t$, $\tilde{\mathbf{v}}_t$ will be time-dependent as parameters $\mathbf{Q}_t^{(ux)}$ and $\mathbf{Q}_t^{(u)}$ depend on time via their dependence on signals \mathbf{z}_t , see Eqs. (10.135).

For a given time step t , the G-learning algorithm keeps iterating between the policy optimization step that updates policy parameters according to Eq. (10.144)

for fixed coefficients of the F - and G -functions, and the policy evaluation step that involves Eqs. (10.134, 10.135, 10.141) and solves for parameters of the F - and G -functions given policy parameters.

At convergence of this iteration for time step t , Eqs. (10.134, 10.135, 10.141) and (10.143) together solve one step of G-learning for the entropy-regularized linear dynamics. The calculation then proceeds by moving to the previous step $t \rightarrow t - 1$, and repeating the calculation. To accelerate convergence, optimal parameters of the policy at time t can be used as parameters of a prior distribution for time step $t - 1$. The whole procedure is then continued all the way back to the present time. Note that as parameters in Eq. (10.141) depend on the signals \mathbf{z}_t , their expected next-step values will have to be computed as indicated in Eq. (10.133).

? Multiple Choice Question 4

Select all the following correct statements:

- a. In G-learning, the conventional action-value function and value function are recovered from F- and G-functions in the limit $\beta \rightarrow 0$.
 - b. In G-learning, the conventional action-value function and value function are recovered from F- and G-functions in the limit $\beta \rightarrow \infty$.
 - c. Linearization of portfolio dynamics with G-learning is needed to get a locally linear G-function and F-function.
 - d. Linearization of portfolio dynamics with G-learning is needed to get a locally quadratic G-function and F-function.
-

5.16 Non-zero Market Impact: Non-linear Dynamics

As we just demonstrated, in the limit of vanishing market impact parameters, dynamic portfolio optimization using G-learning with quadratic rewards and Gaussian reference policies amounts to a probabilistic version of a linear quadratic regulator (LQR) and provides a convenient and fast semi-analytical calculation method to optimize an investment policy for a multi-asset and multi-period portfolio. This provides a probabilistic and multi-period RL-based generalization of the classical Markowitz portfolio optimization problem.

If we turn the market impact parameters μ_i on, this leads to drastic changes in the problem: it is no longer analytically tractable due to non-linearity of dynamics for $\mu_i > 0$. The state equation in this case is

$$\mathbf{x}_{t+1} = (1 + r_f)(\mathbf{x}_t + \mathbf{u}_t) + \text{diag}(\mathbf{W}\mathbf{z}_t - \mathbf{M}\mathbf{u}_t)(\mathbf{x}_t + \mathbf{u}_t) + \varepsilon(\mathbf{x}_t, \mathbf{u}_t),$$

where $\mathbf{M} = \text{diag}(\mu_i)$ and $\varepsilon(\mathbf{x}_t, \mathbf{u}_t) := \varepsilon_t \circ (\mathbf{x}_t + \mathbf{u}_t)$ (see Eqs. (10.123), (10.124)).

When dynamics are non-linear, one possibility is to iteratively linearize the dynamics, similar to the working of an extended Kalman filter. For problems with deterministic policies, an iterative quadratic regulator (IQR) can be used to linearize the dynamics around a reference path at each step of a policy iteration (Todorov and Li 2005). Other methods can be applied when working with stochastic policies. In particular, Halperin and Feldshteyn (2018) explore a variational EM algorithm where Gaussian random hidden variables are used to linearize dynamics, providing a probabilistic version of the IQR. Such approaches, needed when the dynamics are non-linear, are more technically involved than the linear LQR case, and the reader is referred to the literature for details.

Beyond and above of purely computational issues, non-linearity of dynamics leads to important theoretical implications. The problem of optimal control is to find the optimal action \mathbf{u}_t^* as a function of the state \mathbf{x}_t . When this is done, the resulting expression can be substituted back to obtain non-linear open-loop dynamics (i.e., the dynamics where action variables are substituted by their optimal values). As will be discussed later in this book, the ensuing non-linearity of dynamics might have important consequences for capturing the behavior of real markets.

6 RL for Wealth Management

6.1 *The Merton Consumption Problem*

Our two previous use cases for reinforcement learning in quantitative finance, namely option pricing and dynamic portfolio optimization, operate with self-financing portfolios. Recall that self-financing portfolios are asset portfolios that do not admit cash injections or withdrawals at any point in a lifetime of a portfolio, except at its inception and closing. For these classical financial problems, the assumption of a self-financing portfolio is well suited and reasonably matches actual financial practices.

There is yet another wide class of classical problems in finance where a self-financing portfolio is not a good starting point for modeling. Financial planning and wealth management are two good examples of such problems. Indeed, a typical investor in a retirement plan makes periodic investments in the portfolio while being employed and periodically draws from the account when in retirement. In addition to adding or withdrawing capital to the portfolio, the investor can also re-balance the portfolio by selling and buying different assets (e.g., stocks).

The problem of optimal consumption with an investment portfolio is frequently referred to as the *Merton consumption problem*, after the celebrated work of Robert Merton who considered this problem as a problem of continuous-time optimal control with log-normal dynamics for asset prices (Merton 1971). As optimization in problems involving cash injections instead of cash withdrawals formally corresponds to a sign change of one-step consumption in the Merton formulation,

we can collectively refer to all types of wealth management problems involving injections or withdrawals of funds at intermediate time steps as a generalized Merton consumption problem. We shall begin with a simple example which involves a combination of asset allocation and consumption under a specific choice of utility of wealth and lifetime consumption.

Example 10.9 Discrete time Merton consumption with CRRA utility

To illustrate the basic setup, let us consider the problem of allocating wealth between a risky asset and a risk-free asset, in addition to wealth consumption. The optimal consumption problem is formulated in a discrete-time, finite-horizon setting rather than the classical continuous-time approach of Merton (1971). Following Cheung and Yang (2007), we will assume that the investment horizon $T \in \mathcal{N}$ is fixed. We further assume that at the beginning of each time period, an investor can decide the allocation of wealth between the risky asset and the level of consumption, which should be non-negative and less than her total wealth at that time.

Denote the wealth of the investor at time t as W_t , and the random return from the risk assets in time period $[t, t+1]$ is denoted as R_t . The time t consumption level is denoted as $c_t \in [0, W_t]$. After consumption, a proportion $\alpha_t \in [0, 1]$ of the remaining amount will be invested in the risky asset and the rest in the risk-free asset. We refer to these constraints as the “budget constraints.” The discrete-time wealth evolution equation is given by

$$W_{t+1} = (W_t - c_t)[(1 - \alpha_t)R\Delta t + \alpha_t R_f \Delta t]. \quad (10.145)$$

with an initial positive wealth W_0 at time $t = 0$. The sequence of maps $(C, \alpha) = \{(c_0, \alpha_0), \dots, (c_{T-1}, \alpha_{T-1})\}$ that satisfies the budget constraints is called the “investment-consumption strategy.”

The expected sum of the rewards for consumption (i.e., a utility of consumption) with a terminal reward is used as a criterion to measure the performance of an investment-consumption strategy. In RL, we are free to choose any reward function which is concave in the actions. Writing down the optimization problem:

$$\max_{(c_0, \alpha_0), \dots, (C_{T-1}, \alpha_{T-1})} \mathbb{E}[\sum_{t=0}^{T-1} \gamma^t R(W_t, (c_t, \alpha_t), W_{t+1}) + \gamma^T R(W_T) | W_0 = w], \quad (10.146)$$

(continued)

Example 10.9 (continued)

the optimal investment strategy is given by

$$V_t(w) = \max_{(c_t, \alpha_t), \dots, (C_{T-1}, \alpha_{T-1})} \mathbb{E} \left[\sum_{s=t}^{T-1} \gamma^{s-t} R(W_s, (c_s, \alpha_s), W_{s+1}) + \gamma^{T-t} R(W_T) | W_0 = w \right], \quad (10.147)$$

which can be solved from the Bellman Equation for the value function updates:

$$V_t(w) = \max_{(c_t, \alpha_t)} \mathbb{E}[V_{t+1}(W_{t+1}) | W_t = w], \forall t \in \{0, \dots, T-1\}, \quad (10.148)$$

and some terminal condition for $V_T(w)$. A common choice of utility function, which leads to closed-form solutions, is to choose a constant relative risk aversion (CRRA) utility function of the form $U(x) = \frac{1}{\gamma'} x^{\gamma'}$, with $\gamma' \in (0, 1)$. Then the state function reduces to

$$V_t(w) = \frac{w^{\gamma'}}{\gamma'} \left[1 + (\gamma Y_t H_t)^{1/(1-\gamma')} \right]^{1-\gamma'}, \quad (10.149)$$

with optimal consumption, linear in the wealth:

$$\hat{c}_t(w) = \frac{w}{(1 + (\gamma Y_t H_t)^{1/(1-\gamma')})} \leq w, \quad (10.150)$$

where the expected returns of the fund under optimal allocation at time t are

$$Y_t = \mathbb{E}[(\alpha_t^* R_t + (1 - \alpha_t^*) R_f)^{\gamma'}] \quad (10.151)$$

and the recurrent variable is given by the recursion relation

$$H_t = 1 + (\gamma Y_{t+1} H_{t+1})^{1/(1-\gamma')}, \quad (10.152)$$

and we assume $H_T = 0$. Note that there is a unique $\alpha_t^* \in [0, 1]$ such that Y_t is maximized. To show that Y_t is concave in α_t^* , we see that

$$\gamma'(\gamma - 1)'(\alpha_t^* \mathbb{E}[R_t] + (1 - \alpha_t^*) R_f)^{\gamma'-2} (\mathbb{E}[R_t] - R_f)^2 \leq 0, \quad (10.153)$$

(continued)

Example 10.9 (continued)

since $\gamma'(\gamma - 1)' < 0$, $(\mathbb{E}[R_t] - R_f)^2 \geq 0$, and $(\alpha_t^* \mathbb{E}[R_t] + (1 - \alpha_t^*) R_f)^{\gamma' - 2} > 0$ for non-negative risk-free rates and average stock returns. In the absence of transaction costs, clearly when the expected return of the risky asset is above R_f , we allocate the wealth to the risky asset and when the expected return is below, we choose to entirely invest in the risk-free account.

We can therefore simplify the optimization problem under the CRRA utility function to solve for the optimal consumption from Eq. (10.150). Figure 10.6 illustrates the optimal consumption under simulated stock prices. The optimal allocation is not shown here but alternates between 0 and 1 depending on whether the mean return of the risk asset is, respectively, above or below the risk-free rate.

The analytic approach of Cheung and Yang (2007) in the above example is limited to the choice of utility function and single asset portfolio. One can solve the same problem with flexibility in the choice of utility functions using the LSPI algorithm, but such an approach does not extend to higher dimensional portfolios. We therefore turn to a G-learning approach which scales to high-dimensional portfolios while providing some flexibility in the choice of utility functions.

In the following section, we will consider a class of wealth management problems: optimization of a defined contribution retirement plan, where cash is injected (rather than withdrawn) at each time step. Instead of relying on a utility of consumption, along the lines of the approach just previously described, we will adopt a more “RL-native” approach by directly specifying one-step rewards. Another difference is that, as in the previous section, we define actions as absolute (dollar-valued) changes of asset positions, instead of defining them in fractional terms, as in the Merton approach. As we will see shortly, this enables a simple transformation into an unconstrained optimization problem and provides a semi-analytical solution for a particular choice of the reward function.

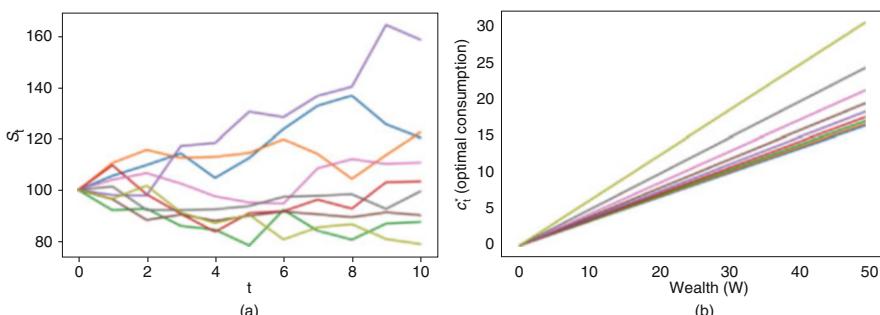


Fig. 10.6 Stock prices are simulated using an Euler scheme over a one-year horizon. At each of ten periods shown by ten separate lines, the optimal consumption is estimated using the closed-form formula in Eq. (10.150). The optimal investment is monotonically increasing in time. **(a)** Simulated stock prices. **(b)** Optimal consumption against wealth

6.2 Portfolio Optimization for a Defined Contribution Retirement Plan

Here we consider a simplified model for retirement planning. We assume a discrete-time process with T steps, so that T is the (integer-valued) time horizon. The investor/planner keeps the wealth in N assets, with \mathbf{x}_t being the vector of dollar values of positions in different assets at time t , and \mathbf{u}_t being the vector of changes in these positions. We assume that the first asset with $n = 1$ is a risk-free bond, and other assets are risky, with uncertain returns \mathbf{r}_t whose expected values are $\bar{\mathbf{r}}_t$. The covariance matrix of return is Σ_r of size $(N - 1) \times (N - 1)$. Note that our notation in this section is different from the previous section where \mathbf{x}_t was used to denote a vector of *risky* asset holding values.

Optimization of a retirement plan involves optimization of both regular contributions to the plan and asset allocations. Let c_t be a cash installment in the plan at time t . The pair (c_t, \mathbf{u}_t) can thus be considered the action variables in a dynamic optimization problem corresponding to the retirement plan.

We assume that at each time step t , there is a pre-specified target value \hat{P}_{t+1} of a portfolio at time $t + 1$. We assume that the target value \hat{P}_{t+1} at step t exceeds the next-step value $V_{t+1} = (1 + \mathbf{r}_t)(\mathbf{x}_t + \mathbf{u}_t)$ of the portfolio, and we want to impose a penalty for under-performance relative to this target. To this end, we can consider the following expected reward for time step t :

$$R_t(\mathbf{x}_t, \mathbf{u}_t, c_t) = -c_t - \lambda \mathbb{E}_t \left[\left(\hat{P}_{t+1} - (1 + \mathbf{r}_t)(\mathbf{x}_t + \mathbf{u}_t) \right)_+ \right] - \mathbf{u}_t^T \boldsymbol{\Omega} \mathbf{u}_t. \quad (10.154)$$

Here the first term is due to an installment of amount c_t at the beginning of time period t , the second term is the expected negative reward from the end of the period for under-performance relative to the target, and the third term approximates transaction costs by a convex functional with the parameter matrix $\boldsymbol{\Omega}$ and serves as a L_2 regularization.

The one-step reward (10.154) is inconvenient to work with due to the rectified non-linearity $(\cdot)_+ := \max(\cdot, 0)$ under the expectation. Another problem is that decision variables c_t and \mathbf{u}_t are not independent but rather satisfy the following constraint:

$$\sum_{n=1}^N u_{tn} = c_t, \quad (10.155)$$

which simply means that at every time step, the total change in all positions should equal the cash installment c_t at this time.

We therefore modify the one-step reward (10.154) in two ways: we replace the first term using Eq. (10.155) and approximate the rectified non-linearity by a quadratic function. The new one-step reward is

$$R_t(\mathbf{x}_t, \mathbf{u}_t) = -\sum_{n=1}^N u_{tn} - \lambda \mathbb{E}_t \left[\left(\hat{P}_{t+1} - (1 + \mathbf{r}_t)(\mathbf{x}_t + \mathbf{u}_t) \right)^2 \right] - \mathbf{u}_t^T \boldsymbol{\Omega} \mathbf{u}_t. \quad (10.156)$$

The new reward function (10.156) is attractive on two counts. First, it explicitly resolves the constraint (10.155) between the cash injection c_t and portfolio allocation decisions, and thus converts the initial constrained optimization problem into an unconstrained one. This differs from the Merton model where allocation variables are defined as fractions of the total wealth, and thus are constrained by construction. The approach based on dollar-measured actions both reduces the dimensionality of the optimization problem and makes it unconstrained. When the unconstrained optimization problem is solved, the optimal contribution c_t at time t can be obtained from Eq. (10.155).

The second attractive feature of the reward (10.156) is that it is quadratic in actions \mathbf{u}_t and is therefore highly tractable. On the other hand, the well-known disadvantage of quadratic rewards (penalties) is that they are symmetric, and penalize both scenarios $V_{t+1} \gg \hat{P}_{t+1}$ and $V_{t+1} \ll \hat{P}_{t+1}$, while in fact we only want to penalize the second class of scenarios. To mitigate this drawback, we can consider target values \hat{P}_{t+1} that are considerably higher than the time- t expectation of the next-period portfolio value. In what follows we assume this is the case, otherwise the value of \hat{P}_{t+1} can be arbitrary. For example, one simple choice could be to set the target portfolio as the current portfolio growing with a fixed and sufficiently high return.

We note that a quadratic loss specification relative to a target time-dependent wealth level is a popular choice in the recent literature on wealth management. One example is provided by Lin et al. (2019) who develop a dynamic optimization approach with a similar squared loss function for a defined contribution retirement plan. A similar approach that relies on a direct specification of a reward based on a target portfolio level is known as “goal-based wealth management” (Browne 1996; Das et al. 2018).

> Goal-Based Wealth Management

Mean–variance Markowitz optimization remains one of the most commonly used tools in wealth management. Portfolio objectives in this approach are defined in terms of expected returns and covariances of asset in the portfolio, which may not be the most natural formulation for retail investors. Indeed, the latter typically seek specific financial goals for their portfolios. For example, a contributor to a retirement plan may demand that the value of their portfolio

(continued)

at the age of his or her retirement be at least equal to, or preferably larger than, some target value P_T .

Goal-based wealth management offers some interesting perspectives into optimal structuring of wealth management plans such as retirement plans or target date funds. The motivation for operating in terms of wealth goals can be more intuitive (while still tractable) than the classical formulation in terms of expected excess returns and variances. To see this, let V_T be the final wealth in the portfolio, and P_T be a certain target wealth level at the horizon T . The goal-based wealth management approach of Browne (1996) and Das et al. (2018) uses the probability $\mathbf{P}[V_T - P_T \geq 0]$ of final wealth V_T to be above the target level P_T as an objective for maximization by an active portfolio management. This probability is the same as the price of a binary option on the terminal wealth V_T with strike P_T : $\mathbf{P}[V_T - P_T \geq 0] = \mathbb{E}_t [\mathbb{1}_{V_T > P_T}]$. Instead of a utility of wealth such as a power or logarithmic utility, this approach uses the price of this binary option as the objective function. This idea can also be modified by using a call option-like expectation $\mathbb{E}_t [(V_T - P_T)_+]$, instead of a binary option. Such an expectation quantifies how much the terminal wealth is expected to exceed the target, rather than simply providing the probability of such an event.

The square loss reward specification is very convenient, as we have already seen on many occasions in this chapter, as it allows one to construct optimal policies semi-analytically. Here we will show how to build a semi-analytical scheme for computing optimal stochastic consumption-investment policies for a retirement plan—the method is sufficiently general for either a cumulation or de-cumulation phase. For other specifications of rewards, numerical optimization and function approximations (e.g., neural networks) would be required.

The expected reward (10.156) can be written in a more explicit form if we denote asset returns as $\mathbf{r}_t = \bar{\mathbf{r}}_t + \tilde{\boldsymbol{\varepsilon}}_t$ where the first component $\bar{r}_0(t) = r_f$ is the risk-free rate (as the first asset is risk-free), and $\tilde{\boldsymbol{\varepsilon}}_t = (0, \boldsymbol{\varepsilon}_t)$, where $\boldsymbol{\varepsilon}_t$ is an idiosyncratic noise with covariance $\boldsymbol{\Sigma}_r$ of size $(N - 1) \times (N - 1)$. Substituting this expression in Eq. (10.156), we obtain

$$\begin{aligned} R_t(\mathbf{x}_t, \mathbf{u}_t) = & -\lambda \hat{P}_{t+1}^2 - \mathbf{u}_t^T \mathbb{1} + 2\lambda \hat{P}_{t+1} (\mathbf{x}_t + \mathbf{u}_t)^T (1 + \bar{\mathbf{r}}_t) \\ & - \lambda (\mathbf{x}_t + \mathbf{u}_t)^T \hat{\boldsymbol{\Sigma}}_t (\mathbf{x}_t + \mathbf{u}_t) - \mathbf{u}_t^T \boldsymbol{\Omega} \mathbf{u}_t, \end{aligned} \quad (10.157)$$

where we defined

$$\hat{\boldsymbol{\Sigma}}_t = \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_r \end{bmatrix} + (1 + \bar{\mathbf{r}}_t)(1 + \bar{\mathbf{r}}_t)^T. \quad (10.158)$$

The quadratic one-step reward (10.157) has a similar structure to the rewards we considered in the previous section, see, e.g., Eq. (10.128). In contrast to the setting in Sect. 5.15, instead of a self-financing portfolio, here we deal with a portfolio with periodic cash installments c_t . However, because the latter are related to allocation decision variables by the constraint (10.155), the resulting quadratic reward (10.157) has the same quadratic structure as the linear LQR reward (10.128).

6.3 G-Learning for Retirement Plan Optimization

As we have just mentioned, the quadratic one-step reward (10.157) is very similar to the reward Eq. (10.128) which we considered in Sect. 5.15 for a self-financing portfolio. The main difference is the presence of the first term $-\lambda \hat{P}_{t+1}^2$ in (10.157) which is independent of a state or action. This term does not impact the policy optimization task, and can be trivially accounted for, if needed, e.g., to compute the total reward, by a direct summation from all time steps in the plan lifeline.

As in Sect. 5.15, we use a similar semi-analytical formulation of G-learning with Gaussian time-varying policies (GTVP). We start by specifying a functional form of the value function as a quadratic form of \mathbf{x}_t :

$$F_t^\pi(\mathbf{x}_t) = \mathbf{x}_t^T \mathbf{F}_t^{(xx)} \mathbf{x}_t + \mathbf{x}_t^T \mathbf{F}_t^{(x)} + F_t^{(0)}, \quad (10.159)$$

where $\mathbf{F}_t^{(xx)}$, $\mathbf{F}_t^{(x)}$, $F_t^{(0)}$ are parameters that can depend on time via their dependence on the target values \hat{P}_{t+1} and the expected returns $\bar{\mathbf{r}}_t$ (in the formulation in Sect. 5.15, the latter were encoded in signals \mathbf{z}_t). The dynamic equation now reads (compare with Eq. (10.126))

$$\mathbf{x}_{t+1} = \mathbf{A}_t (\mathbf{x}_t + \mathbf{u}_t) + (\mathbf{x}_t + \mathbf{u}_t) \circ \tilde{\boldsymbol{\epsilon}}_t, \quad \mathbf{A}_t := \text{diag}(1 + \bar{\mathbf{r}}_t), \quad \tilde{\boldsymbol{\epsilon}}_t := (0, \boldsymbol{\epsilon}_t) \quad (10.160)$$

Coefficients of the value function (10.159) are computed backward in time starting from the last maturity $t = T - 1$. For $t = T - 1$, the quadratic reward (10.157) can be optimized analytically by the following action:

$$\mathbf{u}_{T-1} = \tilde{\boldsymbol{\Sigma}}_{T-1}^{-1} \left(\tilde{\mathbf{P}}_T - \hat{\boldsymbol{\Sigma}}_{T-1} \mathbf{x}_{T-1} \right), \quad (10.161)$$

where we defined parameters $\tilde{\boldsymbol{\Sigma}}_{T-1}$ and $\tilde{\mathbf{P}}_T$ as follows:

$$\tilde{\boldsymbol{\Sigma}}_{T-1} := \hat{\boldsymbol{\Sigma}}_{T-1} + \frac{1}{\lambda} \boldsymbol{\Omega}, \quad \tilde{\mathbf{P}}_T := \hat{P}_T (1 + \bar{\mathbf{r}}_{T-1}) - \frac{1}{2\lambda}. \quad (10.162)$$

Note that the optimal action is a linear function of the state, as in our previous section. Another interesting point to note is that the last term $\sim \boldsymbol{\Omega}$ that describes

convex transaction costs in Eq. (10.157) produces regularization of matrix inversion in Eq. (10.161).

As for the last time step we have $F_{T-1}^\pi(\mathbf{x}_{T-1}) = \hat{R}_{T-1}$, coefficients $\mathbf{F}_{T-1}^{(xx)}$, $\mathbf{F}_{T-1}^{(x)}$, $F_{T-1}^{(0)}$ can be computed by plugging Eq. (10.161) back in Eq. (10.157), and comparing the result with Eq. (10.159) with $t = T - 1$. This provides terminal conditions for parameters in Eq. (10.159):

$$\begin{aligned}\mathbf{F}_{T-1}^{(xx)} &= -\lambda \Theta_{T-1}^T \hat{\Sigma}_{T-1} \Theta_{T-1} - \hat{\Sigma}_{T-1} \tilde{\Sigma}_{T-1}^{-1} \Omega \tilde{\Sigma}_{T-1}^{-1} \hat{\Sigma}_{T-1} \\ \mathbf{F}_{T-1}^{(x)} &= \hat{\Sigma}_{T-1} \tilde{\Sigma}_{T-1}^{-1} \mathbb{1} + 2\lambda \hat{P}_T \Theta_{T-1}^T (1 + \bar{\mathbf{r}}_{T-1}) \\ &\quad - 2\lambda \Theta_{T-1}^T \hat{\Sigma}_{T-1} \tilde{\Sigma}_{T-1}^{-1} \tilde{\mathbf{P}}_T + 2\hat{\Sigma}_{T-1} \tilde{\Sigma}_{T-1}^{-1} \Omega \tilde{\Sigma}_{T-1}^{-1} \tilde{\mathbf{P}}_T \\ F_{T-1}^{(0)} &= -\lambda \hat{P}_T^2 - \tilde{\mathbf{P}}_T^T \tilde{\Sigma}_{T-1}^{-1} \mathbb{1} + 2\lambda \hat{P}_T (1 + \bar{\mathbf{r}}_{T-1})^T \tilde{\Sigma}_{T-1}^{-1} \tilde{\mathbf{P}}_T \\ &\quad - \lambda \tilde{\mathbf{P}}_T^T \tilde{\Sigma}_{T-1}^{-1} \hat{\Sigma}_{T-1} \tilde{\Sigma}_{T-1}^{-1} \tilde{\mathbf{P}}_T - \tilde{\mathbf{P}}_T^T \tilde{\Sigma}_{T-1}^{-1} \Omega \tilde{\Sigma}_{T-1}^{-1} \tilde{\mathbf{P}}_T,.\end{aligned}\quad (10.163)$$

where we defined $\Theta_{T-1} := \mathbb{I} - \tilde{\Sigma}_{T-1}^{-1} \hat{\Sigma}_{T-1}$. For an arbitrary time step $t = T - 2, \dots, 0$, we use Eq. (10.160) to compute the conditional expectation of the next-period F-function in the Bellman equation (10.115) as follows:

$$\begin{aligned}\mathbb{E}_{t,\mathbf{a}}[F_{t+1}^\pi(\mathbf{x}_{t+1})] &= (\mathbf{x}_t + \mathbf{u}_t)^T \left(\mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(xx)} \mathbf{A}_t + \tilde{\Sigma}_r \circ \bar{\mathbf{F}}_{t+1}^{(xx)} \right) (\mathbf{x}_t + \mathbf{u}_t) \\ &\quad + (\mathbf{x}_t + \mathbf{u}_t)^T \mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(x)} + \bar{F}_{t+1}^{(0)}, \quad \tilde{\Sigma}_r := \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \Sigma_r \end{bmatrix},\end{aligned}\quad (10.164)$$

where $\bar{\mathbf{F}}_{t+1}^{(xx)} := \mathbb{E}_t[\mathbf{F}_{t+1}^{(xx)}]$, and similarly for $\bar{\mathbf{F}}_{t+1}^{(x)}$ and $\bar{F}_{t+1}^{(0)}$. This is a quadratic function of \mathbf{x}_t and \mathbf{u}_t and has the same structure as the quadratic reward $\hat{R}(\mathbf{x}_t, \mathbf{a}_t)$ in Eq. (10.157). Plugging both expressions in the Bellman equation

$$G_t^\pi(\mathbf{y}_t, \mathbf{a}_t) = \hat{R}_t(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t,\mathbf{a}}[F_{t+1}^\pi(\mathbf{y}_{t+1}) | \mathbf{y}_t, \mathbf{a}_t]$$

we see that the action-value function $G_t^\pi(\mathbf{x}_t, \mathbf{u}_t)$ should also be a quadratic function of \mathbf{x}_t and \mathbf{u}_t :

$$G_t^\pi(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^T \mathbf{Q}_t^{(xx)} \mathbf{x}_t + \mathbf{x}_t^T \mathbf{Q}_t^{(xu)} \mathbf{u}_t + \mathbf{u}_t^T \mathbf{Q}_t^{(uu)} \mathbf{u}_t + \mathbf{x}_t^T \mathbf{Q}_t^{(x)} + \mathbf{u}_t^T \mathbf{Q}_t^{(u)} + Q_t^{(0)}, \quad (10.165)$$

where

$$\begin{aligned}\mathbf{Q}_t^{(xx)} &= -\lambda \hat{\Sigma}_t + \gamma \left(\mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(xx)} \mathbf{A}_t + \tilde{\Sigma}_r \circ \bar{\mathbf{F}}_{t+1}^{(xx)} \right) \\ \mathbf{Q}_t^{(xu)} &= 2\mathbf{Q}_t^{(xx)} \\ \mathbf{Q}_t^{(uu)} &= \mathbf{Q}_t^{(xx)} - \Omega\end{aligned}\quad (10.166)$$

$$\begin{aligned}\mathbf{Q}_t^{(x)} &= 2\lambda \hat{P}_{t+1}(1 + \bar{\mathbf{r}}_t) + \gamma \mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(x)} \\ \mathbf{Q}_t^{(u)} &= \mathbf{Q}_t^{(x)} - \mathbb{1} \\ Q_t^{(0)} &= -\lambda \hat{P}_{t+1}^2 + \gamma F_{t+1}^{(0)}.\end{aligned}$$

Note that the quadratic action-value function in Eq.(10.165) is similar to Eq.(10.134)—the only difference is the specification of the parameters.

Beyond different expressions for coefficients of the value function $F_t(x_t)$ and action-value function $G_t(x_t, u_t)$ and a different terminal condition, the rest of calculations to perform one step of G-learning is the same as in Sect. 5.15. The F-function for the current step can be found using Eq.(10.136) repeated again here:

$$F_t^\pi(\mathbf{x}_t) = \frac{1}{\beta} \log \int \pi_0(\mathbf{u}_t | \mathbf{x}_t) e^{\beta G_t^\pi(\mathbf{x}_t, \mathbf{u}_t)} d\mathbf{u}_t. \quad (10.167)$$

A reference policy $\pi_0(\mathbf{u}_t | \mathbf{x}_t)$ is Gaussian:

$$\pi_0(\mathbf{u}_t | \mathbf{x}_t) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_p|}} e^{-\frac{1}{2} (\mathbf{u}_t - \hat{\mathbf{u}}_t)^T \Sigma_p^{-1} (\mathbf{u}_t - \hat{\mathbf{u}}_t)}, \quad (10.168)$$

where the mean value $\hat{\mathbf{u}}_t$ is a linear function of the state \mathbf{x}_t :

$$\hat{\mathbf{u}}_t = \bar{\mathbf{u}}_t + \bar{\mathbf{v}}_t \mathbf{x}_t. \quad (10.169)$$

Again as in Sect. 5.15, integration over \mathbf{u}_t in Eq.(10.167) is performed analytically using Eq.(10.139). The difference is that in Sect. 5.15 we considered a self-financing asset portfolio, which constrains actions \mathbf{u}_t . Ignoring such a constraint can produce numerical inaccuracies. In contrast, in the present case we do not impose constraints on actions \mathbf{u}_t ; therefore, an unconstrained multivariate Gaussian integration should be superior in this case. Remarkably, this implies that once the decision variables are chosen appropriately, portfolio optimization for wealth management tasks may in a sense be an easier problem than portfolio optimization with self-financing.

Performing the Gaussian integration and comparing the resulting expression with Eq.(10.159), we obtain for its coefficients:

$$\begin{aligned}F_t^\pi(\mathbf{x}_t) &= \mathbf{x}_t^T \mathbf{F}_t^{(xx)} \mathbf{x}_t + \mathbf{x}_t^T \mathbf{F}_t^{(x)} + F_t^{(0)} \\ \mathbf{F}_t^{(xx)} &= \mathbf{Q}_t^{(xx)} + \frac{1}{2\beta} \left(\mathbf{U}_t^T \bar{\Sigma}_p^{-1} \mathbf{U}_t - \bar{\mathbf{v}}_t^T \Sigma_p^{-1} \bar{\mathbf{v}}_t \right) \\ \mathbf{F}_t^{(x)} &= \mathbf{Q}_t^{(x)} + \frac{1}{\beta} \left(\mathbf{U}_t^T \bar{\Sigma}_p^{-1} \mathbf{W}_t - \bar{\mathbf{v}}_t^T \Sigma_p^{-1} \bar{\mathbf{u}}_t \right) \\ \mathbf{F}_t^{(0)} &= \mathbf{Q}_t^{(0)} + \frac{1}{2\beta} \left(\mathbf{W}_t^T \bar{\Sigma}_p^{-1} \mathbf{W}_t - \bar{\mathbf{u}}_t^T \Sigma_p^{-1} \bar{\mathbf{u}}_t \right) - \frac{1}{2\beta} (\log |\Sigma_p| + \log |\bar{\Sigma}_p|),\end{aligned} \quad (10.170)$$

where we use the auxiliary parameters

$$\begin{aligned}\mathbf{U}_t &= \beta \mathbf{Q}_t^{(ux)} + \Sigma_p^{-1} \bar{\mathbf{v}}_t \\ \mathbf{W}_t &= \beta \mathbf{Q}_t^{(u)} + \Sigma_p^{-1} \bar{\mathbf{u}}_t \\ \bar{\Sigma}_p &= \Sigma_p^{-1} - 2\beta \mathbf{Q}_t^{(uu)}.\end{aligned}\quad (10.171)$$

The optimal policy for the given step can be found using Eq. (10.142) repeated again here:

$$\pi(\mathbf{u}_t | \mathbf{x}_t) = \pi_0(\mathbf{u}_t | \mathbf{x}_t) e^{\beta(G_t^\pi(\mathbf{x}_t, \mathbf{u}_t) - F_t^\pi(\mathbf{x}_t))}. \quad (10.172)$$

Using here the quadratic action-value function (10.165) produces a new Gaussian policy $\pi(\mathbf{u}_t | \mathbf{x}_t)$:

$$\pi(\mathbf{u}_t | \mathbf{x}_t) = \frac{1}{\sqrt{(2\pi)^n |\tilde{\Sigma}_p|}} e^{-\frac{1}{2}(\mathbf{u}_t - \tilde{\mathbf{u}}_t - \tilde{\mathbf{v}}_t \mathbf{x}_t)^T \tilde{\Sigma}_p^{-1} (\mathbf{u}_t - \tilde{\mathbf{u}}_t - \tilde{\mathbf{v}}_t \mathbf{x}_t)}, \quad (10.173)$$

where

$$\begin{aligned}\tilde{\Sigma}_p^{-1} &= \Sigma_p^{-1} - 2\beta \mathbf{Q}_t^{(uu)} \\ \tilde{\mathbf{u}}_t &= \tilde{\Sigma}_p \left(\Sigma_p^{-1} \bar{\mathbf{u}}_t + \beta \mathbf{Q}_t^{(u)} \right) \\ \tilde{\mathbf{v}}_t &= \tilde{\Sigma}_p \left(\Sigma_p^{-1} \bar{\mathbf{v}}_t + \beta \mathbf{Q}_t^{(ux)} \right)\end{aligned}\quad (10.174)$$

Therefore, policy optimization for G-learning with quadratic rewards and Gaussian reference policy amounts to the Bayesian update of the prior distribution (10.168) with parameters updates $\bar{\mathbf{u}}_t$, $\bar{\mathbf{v}}_t$, Σ_p to the new values $\tilde{\mathbf{u}}_t$, $\tilde{\mathbf{v}}_t$, $\tilde{\Sigma}_p$ defined in Eqs. (10.174). These quantities depend on time via their dependence on the targets \hat{P}_t and expected asset returns $\bar{\mathbf{r}}_t$.

As in Sect. 5.15, for a given time step t , the G-learning algorithm keeps iterating between the policy optimization step that updates policy parameters according to Eq. (10.174) for fixed coefficients of the F - and G -functions, and the policy evaluation step that involves Eqs. (10.165, 10.166, 10.170) and solves for parameters of the F - and G -functions given policy parameters. Note that convergence of iterations for $\tilde{\mathbf{u}}_t$, $\tilde{\mathbf{v}}_t$ is guaranteed as $|\tilde{\Sigma}_p \Sigma_p^{-1}| < 1$. At convergence of iteration for time step t , Eqs. (10.165, 10.166, 10.170) and (10.143) together solve one step of G-learning. The calculation then proceeds by moving to the previous step $t \rightarrow t-1$, and repeating the calculation, all the way back to the present time.

The additional step needed from G-learning for the present problem is to find the optimal cash contribution for each time step by using the budget constraint (10.155). As G-learning produces Gaussian random actions \mathbf{u}_t , Eq. (10.155) implies that the time- t optimal contribution c_t is Gaussian distributed with mean $\bar{c}_t = \mathbb{1}^T (\bar{\mathbf{u}}_t + \bar{\mathbf{v}}_t \mathbf{x}_t)$. The expected optimal contribution \bar{c}_t thus has a part $\sim \bar{\mathbf{u}}_t$ that is independent of the portfolio value, and a part $\sim \bar{\mathbf{v}}_t$ that depends on the current portfolio. This is similar, e.g., to a linear specification of the defined contribution with a deterministic policy in Lin et al. (2019).

It should be noted that in practice, we may want to impose some constraints on cash installments c_t . For example, we could impose band constraints $0 \leq c_t \leq c_{max}$ with some upper bound c_{max} . Such constraints can be easily added to the framework. To this end, we need to replace the exactly solvable unconstrained least squares problem with a constrained least squares problem. This can be done without a substantial increase of computational time using efficient off-the-shell convex optimization software. An illustration of an optimal solution trajectory obtained without enforcing any constraints is shown in Fig. 10.7 which presents simulation results for a portfolio of 100 assets with 30 time steps. For the specific choice of model parameters used in this example, the model optimally chooses to invest approximately equal contributions around \$500 that slightly increase towards the end of the plan without enforcing constraints, which is achieved by setting a high target portfolio. However, in a more general setting adding constraints might be desirable.

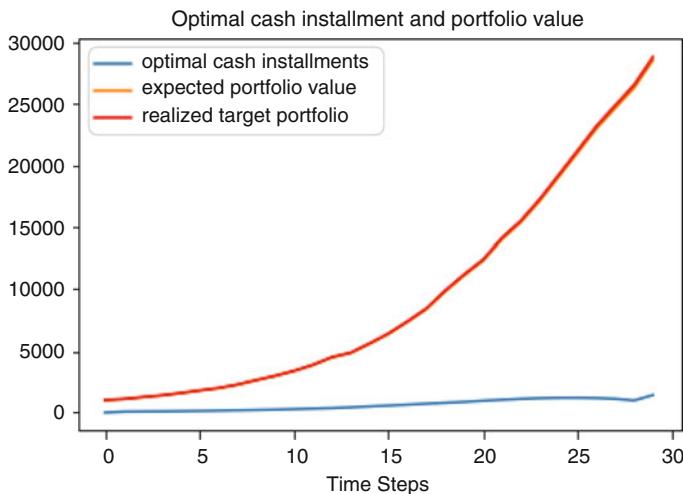


Fig. 10.7 An illustration of G-learning with Gaussian time-varying policies (GTVP) for a retirement plan optimization using a portfolio with 100 assets

6.4 Discussion

To summarize, we have shown how the same G-learning method with quadratic rewards that we used in the previous section to construct an optimal policy for dynamic portfolio optimization can also be used for wealth management problems, provided we use absolute (dollar-nominated) asset position changes as action variables and choose a reward function which is quadratic in these actions. As shown in Sect. 5.15, we found that G-learning applied in the current setting with a quadratic reward and Gaussian reference policy gives rise to an entropy-regulated LQR as a tool for wealth management tasks. This approach results in a Gaussian optimal policy whose mean is a linear function of the state \mathbf{x}_t .

The method we presented enables extensions to other formulations including constrained versions or other specifications of the reward function. One possibility is to use the definition (10.154) with the constraint (10.155), which provides an example of a non-quadratic concave reward. Such cases should be dealt with using flexible function approximations for the action-value function such as neural networks.

7 Summary

The main underlying idea of this chapter was to show that reinforcement learning provides a very natural framework for some of most classical problems of quantitative finance: option pricing and hedging, portfolio optimization, and wealth management problems. As trading in individual assets (or pairs, or buckets of assets) is a particular case of the general portfolio optimization problem, we can say that this list covers most cases for quantitative trading or planning problems in finance.

We saw that for option pricing with reinforcement learning, batch-mode Q-learning can be used as a way to produce a distribution-free discrete-time approach to pricing and hedging of options. In the simplest case when transaction costs and market impact are neglected, as in the classical Black–Scholes model, the reinforcement learning approach is semi-analytical and only requires solving linear matrix equations. In other cases, for example, if the exponential utility is used to enforce a strict no-arbitrage, analytic quadratic optimization should be replaced by numerical convex optimization.

We then presented a multivariate and probabilistic extension of Q-learning known as G-learning, as a tool for using reinforcement learning for portfolio optimization with multiple assets. Unlike the previous case of the QLBS model that neglects transaction costs and market impact, the G-learning approach captures these effects. When the reward function is quadratic as in the Markowitz mean-variance approach and market impact is neglected, the G-learning approach again yields a semi-analytical solution to the dynamic portfolio optimization, that is given by a probabilistic version of the classical linear quadratic regulator (LQR).

For other cases (e.g., when market impact is incorporated, or when rewards are not quadratic), the G-learning approach should rely on more involved numerical optimization methods and/or use function approximations such as neural networks.

In addition to demonstrating how G-learning can be applied for portfolio management, we also showed how it can be used for tasks of wealth management, which differ from the former case by intermediate cash-flows as additional controls. We showed that both these classical financial problems can be treated using the same computational method (G-learning) with different parameter specifications. Therefore, the reinforcement learning approach is capable of providing a unified approach to these classes of problems, which are traditionally treated as different problems because of a different definition of control variables. What we showed is that when using absolute (dollar-nominated) decision variables, both problems can be treated in the same way. Moreover, with this approach wealth management problems turn out to be simpler, not harder, than traditional portfolio optimization problems, as they amount to unconstrained optimization.⁹

While our presentation in this chapter used general RL methods (Q-learning and G-learning), we mostly focused on cases with quadratic rewards which enable semi-analytical and hence easily understandable computational methods. Different reward specifications are possible, of course, but they require relying on function approximations (e.g., neural networks—giving rise to deep reinforcement learning) and numerical optimization for optimizing parameters of these networks. Furthermore, other methods of reinforcement learning (e.g., LSPI, policy-gradient methods, actor-critic, etc.) can also be used, see, e.g., Sato (2019) for a review.

8 Exercises

Exercise 10.1

Derive Eq. (10.46) that gives the limit of the optimal action in the QLBS model in the continuous-time limit.

Exercise 10.2

Consider the expression (10.121) for optimal policy obtained with G-learning

$$\pi(\mathbf{a}_t | \mathbf{y}_t) = \frac{1}{Z_t} \pi_0(\mathbf{a}_t | \mathbf{y}_t) e^{\hat{R}(\mathbf{y}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}_t} [F_{t+1}^\pi(\mathbf{y}_{t+1})]}$$

where the one-step reward is quadratic as in Eq. (10.91):

$$\hat{R}(\mathbf{y}_t, \mathbf{a}_t) = \mathbf{y}_t^T \mathbf{R}_{yy} \mathbf{y}_t + \mathbf{a}_t^T \mathbf{R}_{aa} \mathbf{a} + \mathbf{a}_t^T \mathbf{R}_{ay} \mathbf{y}_t + \mathbf{a}_t^T \mathbf{R}_a.$$

⁹Or, if we want to put additional constraints on the resulting cash-flows, to optimization with one constraint, instead of two constraints as in the Merton approach.

How does this relation simplify in two cases: (a) when the conditional expectation $\mathbb{E}_{t,\mathbf{a}}[F_{t+1}^\pi(\mathbf{y}_{t+1})]$ does not depend on the action a_t , and (b) when the dynamics are linear in a_t as in Eq. (10.125)?

Exercise 10.3

Derive relations (10.141).

Exercise 10.4

Consider G-learning for a time-stationary case, given by Eq. (10.122):

$$G^\pi(\mathbf{y}, \mathbf{a}) = \hat{R}(\mathbf{y}, \mathbf{a}) + \frac{\gamma}{\beta} \sum_{\mathbf{y}'} \rho(\mathbf{y}'|\mathbf{y}, \mathbf{a}) \log \sum_{\mathbf{a}'} \pi_0(\mathbf{a}'|\mathbf{y}') e^{\beta G^\pi(\mathbf{y}', \mathbf{a}')}$$

Show that the high-temperature limit $\beta \rightarrow 0$ of this equation reproduces the fixed-policy Bellman equation for $G^\pi(\mathbf{y}, \mathbf{a})$ where the policy coincides with the prior policy, i.e. $\pi = \pi_0$.

Exercise 10.5

Consider policy update equations for G-learning given by Eqs. (10.174):

$$\tilde{\Sigma}_p^{-1} = \Sigma_p^{-1} - 2\beta \mathbf{Q}_t^{(uu)}$$

$$\tilde{\mathbf{u}}_t = \tilde{\Sigma}_p \left(\Sigma_p^{-1} \bar{\mathbf{u}}_t + \beta \mathbf{Q}_t^{(u)} \right)$$

$$\tilde{\mathbf{v}}_t = \tilde{\Sigma}_p \left(\Sigma_p^{-1} \bar{\mathbf{v}}_t + \beta \mathbf{Q}_t^{(ux)} \right)$$

- (a) Find the limiting forms of these expressions in the high-temperature limit $\beta \rightarrow 0$ and low-temperature limit $\beta \rightarrow \infty$.
- (b) Assuming that we know the stable point $(\bar{\mathbf{u}}_t, \bar{\mathbf{v}}_t)$ of these iterative equations, as well as the covariance $\tilde{\Sigma}_p$, invert them to find parameters of Q -function in terms of stable point values $\bar{\mathbf{u}}_t, \bar{\mathbf{v}}_t$. Note that only parameters $\mathbf{Q}_t^{(uu)}, \mathbf{Q}_t^{(ux)}$, and $\mathbf{Q}_t^{(u)}$ can be recovered. Can you explain why parameters $\mathbf{Q}_t^{(xx)}$ and $\mathbf{Q}_t^{(x)}$ are lost in this procedure? (Note: this problem can be viewed as a prelude to the topic of inverse reinforcement learning covered in the next chapter.)

Exercise 10.6***

The formula for an unconstrained Gaussian integral in n dimensions reads

$$\int e^{-\frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{B}} d^n \mathbf{x} = \sqrt{\frac{(2\pi)^n}{|\mathbf{A}|}} e^{\frac{1}{2} \mathbf{B}^T \mathbf{A}^{-1} \mathbf{B}}.$$

Show that when a constraint $\sum_{i=1}^n x_i \leq \bar{\mathbf{X}}$ with a parameter $\bar{\mathbf{X}}$ is imposed on the integration variables, a constrained version of this integral reads

$$\int e^{-\frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{B}} \theta\left(\bar{\mathbf{X}} - \sum_{i=1}^n x_i\right) d^n \mathbf{x} = \sqrt{\frac{(2\pi)^n}{|\mathbf{A}|}} e^{\frac{1}{2} \mathbf{B}^T \mathbf{A}^{-1} \mathbf{B}} \left(1 - N\left(\frac{\mathbf{B}^T \mathbf{A}^{-1} \mathbf{1} - \bar{\mathbf{X}}}{\sqrt{\mathbf{1}^T \mathbf{A}^{-1} \mathbf{1}}}\right)\right)$$

where $N(\cdot)$ is the cumulative normal distribution.

Hint: use the integral representation of the Heaviside step function

$$\theta(x) = \lim_{\varepsilon \rightarrow 0} \frac{1}{2\pi i} \int_{-\infty}^{\infty} \frac{e^{izx}}{z - i\varepsilon} dz.$$

Appendix

Answers to Multiple Choice Questions

Question 1

Answer: 2, 3.

Question 2

Answer: 2, 4.

Question 3

Answer: 1, 2, 3.

Question 4

Answer: 2, 4.

Python Notebooks

This chapter is accompanied by two notebooks which implement the QLBS model for option pricing and optimal hedging, and G-learning for wealth management. Further details of the notebooks are included in the README .md file.

References

- Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3), 637–654.
- Boyd, S., Bussetti, E., Diamond, S., Kahn, R., Koh, K., Nystrup, P., et al. (2017). Multi-period trading via convex optimization. *Foundations and Trends in Optimization*, 1–74.
- Browne, S. (1996). Reaching goals by a deadline: digital options and continuous-time active portfolio management. https://www0.gsb.columbia.edu/mygsb/faculty/research/pubfiles/841/sidbrowne_deadlines.pdf.

- Carr, P., Ellis, K., & Gupta, V. (1988). Static hedging of exotic options. *Journal of Finance*, 53(3), 1165–1190.
- Cerný, A., & Kallsen, J. (2007). Hedging by sequential regression revisited. Working paper, City University London and TU München.
- Cheung, K. C., & Yang, H. (2007). Optimal investment-consumption strategy in a discrete-time model with regime switching. *Discrete and Continuous Dynamical Systems*, 8(2), 315–332.
- Das, S. R., Ostrov, D., Radhakrishnan, A., & Srivastav, D. (2018). Dynamic portfolio allocation in goals-based wealth management. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3211951.
- Duan, J. C., & Simonato, J. G. (2001). American option pricing under GARCH by a Markov chain approximation. *Journal of Economic Dynamics and Control*, 25, 1689–1718.
- Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch model reinforcement learning. *Journal of Machine Learning Research*, 6, 405–556.
- Föllmer, H., & Schweizer, M. (1989). Hedging by sequential regression: An introduction to the mathematics of option trading. *ASTIN Bulletin*, 18, 147–160.
- Fox, R., Pakman, A., & Tishby, N. (2015). Taming the noise in reinforcement learning via soft updates. In *32nd Conference on Uncertainty in Artificial Intelligence (UAI)*. <https://arxiv.org/pdf/1512.08562.pdf>.
- Garleanu, N., & Pedersen, L. H. (2013). Dynamic trading with predictable returns and transaction costs. *Journal of Finance*, 68(6), 2309–2340.
- Gosavi, A. (2015). Finite horizon Markov control with one-step variance penalties. In *Conference Proceedings of the Allerton Conferences*, Allerton, IL.
- Grau, A. J. (2007). *Applications of least-square regressions to pricing and hedging of financial derivatives*. PhD. thesis, Technische Universität München.
- Halperin, I. (2018). QLBS: Q-learner in the Black-Scholes(-Merton) worlds. *Journal of Derivatives* 2020, (to be published). Available at https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3087076.
- Halperin, I. (2019). The QLBS Q-learner goes NuQLear: Fitted Q iteration, inverse RL, and option portfolios. *Quantitative Finance*, 19(9). <https://doi.org/10.1080/14697688.2019.1622302>, available at https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3102707.
- Halperin, I., & Feldshteyn, I. (2018). Market self-learning of signals, impact and optimal trading: invisible hand inference with free energy, (or, how we learned to stop worrying and love bounded rationality). https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3174498.
- Lin, C., Zeng, L., & Wu, H. (2019). Multi-period portfolio optimization in a defined contribution pension plan during the decumulation phase. *Journal of Industrial and Management Optimization*, 15(1), 401–427. <https://doi.org/10.3934/jimo.2018059>.
- Longstaff, F. A., & Schwartz, E. S. (2001). Valuing American options by simulation - a simple least-square approach. *The Review of Financial Studies*, 14(1), 113–147.
- Markowitz, H. (1959). *Portfolio selection: efficient diversification of investment*. John Wiley.
- Marschinski, R., Rossi, P., Tavoni, M., & Cocco, F. (2007). Portfolio selection with probabilistic utility. *Annals of Operations Research*, 151(1), 223–239.
- Merton, R. C. (1971). Optimum consumption and portfolio rules in a continuous-time model. *Journal of Economic Theory*, 3(4), 373–413.
- Merton, R. C. (1974). Theory of rational option pricing. *Bell Journal of Economics and Management Science*, 4(1), 141–183.
- Murphy, S. A. (2005). A generalization error for Q-learning. *Journal of Machine Learning Research*, 6, 1073–1097.
- Ortega, P. A., & Lee, D. D. (2014). An adversarial interpretation of information-theoretic bounded rationality. In *Proceedings of the Twenty-Eighth AAAI Conference on AI*. <https://arxiv.org/abs/1404.5668>.
- Petrelli, A., Balachandran, R., Siu, O., Chatterjee, R., Jun, Z., & Kapoor, V. (2010). Optimal dynamic hedging of equity options: residual-risks transaction-costs. *working paper*.
- Potters, M., Bouchaud, J., & Sestovic, D. (2001). Hedged Monte Carlo: low variance derivative pricing with objective probabilities. *Physica A*, 289, 517–525.

- Sato, Y. (2019). Model-free reinforcement learning for financial portfolios: a brief survey. <https://arxiv.org/pdf/1904.04973.pdf>.
- Schweizer, M. (1995). Variance-optimal hedging in discrete time. *Mathematics of Operations Research*, 20, 1–32.
- Todorov, E., & Li, W. (2005). A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceeding of the American Control Conference*, Portland OR, USA, pp. 300–306.
- van Hasselt, H. (2010). Double Q-learning. *Advances in Neural Information Processing Systems*. <http://papers.nips.cc/paper/3964-double-q-learning.pdf>.
- Watkins, C. J. (1989). Learning from delayed rewards. Ph.D. Thesis, Kings College, Cambridge, England.
- Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(179–192), 3–4.
- Wilmott, P. (1998). *Derivatives: the theory and practice of financial engineering*. Wiley.

Chapter 11

Inverse Reinforcement Learning and Imitation Learning



This chapter provides an overview of the most popular methods of inverse reinforcement learning (IRL) and imitation learning (IL). These methods solve the problem of optimal control in a data-driven way, similarly to reinforcement learning, however with the critical difference that now rewards are *not* observed. The problem is rather to learn the reward function from the observed behavior of an agent. As behavioral data without rewards is widely available, the problem of learning from such data is certainly very interesting. This chapter provides a moderate-level technical description of the most promising IRL methods, equips the reader with sufficient knowledge to understand and follow the current literature on IRL, and presents examples that use simple simulated environments to evaluate how these methods perform when the “ground-truth” rewards are known. We then present use cases for IRL in quantitative finance which include applications in trading strategy identification, sentiment-based trading, option pricing, inference of portfolio investors, and market modeling.

1 Introduction

One of the challenges faced by researchers when applying reinforcement learning to solve real-world problems is how to choose the reward function. In general, a reward function should encourage a desired behavior of an agent, but often there are multiple approaches to specify it. For example, assume that we seek to solve an index tracking problem, where the task is to replicate a certain market index (e.g., the S&P 500) P_t^{target} by a smaller portfolio of stocks whose value at time t is P_t^{track} . We can view expressions $|P_t^{track} - P_t^{target}|$ or $(P_t^{track} - P_t^{target})^2$, or $(P_t^{track} - P_t^{target})_+$ as possible reward functions. Clearly the optimal choice of the reward here is equivalent to the optimal choice of an expected risk-adjusted return

in the corresponding multi-period portfolio optimization problem. Therefore, the choice of a reward function is as non-unique as the choice of a risk function for portfolio optimization.

Challenges of defining a good reward function to teach an agent to perform a certain task are well known in other fields that use machine learning. For example, teaching physical robots to perform simple (for humans) tasks such as carrying a cup of coffee between two tables by hand-engineering a reward function in a multi-dimensional space of robot joints' positions and velocities at each time step may be as difficult as defining the execution policy directly, without relying on any reward function. Therefore the need to pre-specify a reward function considerably limits the applicability of reinforcement learning for many cases of practical interest. In finance, traders do not often think in terms of any specific utility (reward) function, but rather in terms of strategies (or policies, using the language of RL).

In response to such practical challenges, researchers in machine learning developed a number of alternatives to the classical setting of dynamic programming and reinforcement learning that do *not* require a reward (utility) function to be specified. Learning to act without a reward function is known as *learning from demonstrations* or learning from behavior. As behavioral data is often produced in abundance (think of GPS monitoring data, cell phone data, web browsing data, etc.), the notion of learning from observed behavior of other agents (humans or machines) is certainly appealing, and has a wide range of potential industrial and business applications.

But what exactly does it mean to learn from demonstrations? One possible answer to this question is that it means learning the optimal policy from the observed behavior given only observations of actions produced by this policy (or a similar one). This is called *imitation learning*. This is similar to the batch-mode RL that we considered in the previous chapter, but without knowledge of the rewards. That is, we observe a series of states and actions taken by an agent, and our task is to find the optimal policy solely from this data. As with batch-mode RL, this is an inference problem of a distribution (policy) from data. In contrast to batch-mode RL, however, the problem of learning from demonstrations is ill-defined. Indeed for any particular trajectory of states and actions, there exist an infinite number of policies consistent with this trajectory. The same holds for any finite set of observed trajectories. The problem of learning a policy from a finite set of observed trajectories is therefore an *ill-posed inverse problem*.

> III-Posed Inverse Problems

Ill-posed inverse problems usually exhibit an infinite number of solutions or no solution at all. Classical examples of such inverse problems include the problems of restoring a signal after passing a filter and contaminated by

(continued)

an additive noise. A simple financial example is implying the risk-neutral distribution $p(s_T|s_0)$ of future stock prices from observed prices of European options. If $F(s_T, K)$ is a discounted payoff of a liquid European option with maturity T , the observed market mid-price of the option can be written as

$$C(s_t, K) = \int ds_t p(s_T|s_0)F(s_T, K) + \varepsilon_t,$$

where ε_t stands for an observation noise. Having market prices for a finite number of quoted options is not sufficient to reconstruct a conditional density $p(s_T|s_0)$ in a model-independent way, as a model-independent specification is equivalent to an infinite number of observations. The inverse problem is ill-posed in the sense that it does not have a unique solution. Various forms of regularization are needed in order to select the “best” solution. For example, one may use Lagrange multipliers to enforce constraints on the entropy of $p(s_T|s_0)$ or its KL divergence with some reference (“prior”¹). What constitutes the “best” solution of the inverse problem is therefore only specified after a regularization function is chosen. A good choice of a regularization function may be problem- or domain-specific and go beyond a KL regularization or, e.g., a simple L_2 or L_1 regularization. Essentially, as the choice of regularization amounts to the choice of a “regularization model,” we may assert that a purely model-independent method for inverse problems does not exist.

We may be tempted to reason that such a problem is scarcely different from the conventional setting of supervised learning. Indeed, we could treat the observed states and actions in a training dataset as, respectively, features and outputs. We could then try to directly construct a policy as either a classifier or a regressor, considering each action as a separate observation.

This approach is indeed possible and is known as *behavioral cloning*. While it is simple (any supervised method can be used), it also has a number of serious drawbacks that render it impractical. The main problem is that it often does not generalize well to new unseen states. This is simply because as an action policy is estimated from each individual state using supervised learning, it passes no information on how these states can be related to each other. This can be contrasted with TD methods (such as Q-learning) that use transitions as observations. As

¹It is convenient to regard the reference distribution as a prior, however it is not strictly a prior in the context of Bayesian estimation. MaxEnt or Minimum Cross-Entropy finds a distribution that is compatible with all available constraints and minimizes a KL distance to this reference distribution. In contrast, Bayesian learning involves using Bayes’ rule to incrementally update the posterior.

a result, generalization of the policy to unseen states obtained using supervised learning with behavioral cloning loses any connection to the actual dynamics of the environment. If such a learned policy is executed over multiple steps, errors can compound, and an induced state distribution can shift away from the actual one used in demonstrations.

Therefore, a combination of different methods is usually required when rewards are not available. Such a multi-faceted approach sounds reasonable in principle but the devil is in the detail. For example, we could use a recurrent neural network to capture the state dynamics, and use a feedforward network to directly parameterize the policy. Parameters of both networks would then be learned from the data using, e.g., stochastic gradient descent methods.

The main potential problem with such an approach is that it may not be easily portable to other environments, when the model is used with dynamics that are different from those used for model training. But in an approach similar to the one described above, dynamics and the learned policy are intertwined in complex ways. Therefore, we can expect that a learned policy would become sub-optimal once the dynamics (environment) changes.

On the other hand, a reward function *is* portable, as it depends only on states and actions, but is not concerned with how these states are reached. If we find a way to learn the reward function from demonstrated behavior, this function would be portable to other environments, as it expresses properties of the agent but not of the environment. The idea of learning the reward function as a condensed and portable representation of an agent's preferences was suggested by Russell (1998). Methods centered around this idea have collectively became known as inverse reinforcement learning (IRL). Clearly, if the reward function is found from IRL, then the optimal policy with any new environment can be found using the conventional (direct) RL.

In this chapter, we provide an overview of the most popular methods of IRL, as well as methods of imitation learning that do not rely on a learned reward function. We hasten to add that so far, IRL was only adopted for financial applications in a handful of publications, despite several successful applications in robotics and video games. Nevertheless, we believe that methods of IRL are potentially very useful for quantitative finance.

Both because the whole field of IRL is still nascent and keeps evolving, and because there are only a few published papers on using IRL for financial applications, this chapter is mostly focused on theoretical concepts of IRL. Our task in this chapter is three-fold: (i) provide a reasonably high-level description of the most promising IRL methods; (ii) equip the reader with enough knowledge to understand and follow the current literature on IRL; and (iii) present use cases for IRL in quantitative finance including applications to trading strategy identification, sentiment-based trading, option pricing, inference of portfolio investors, and market modeling.

Chapter Objectives

This chapter will review some of the most pertinent aspects of inverse reinforcement learning and their application to finance:

- Introduce methods of inverse reinforcement learning (IRL) and imitation learning (IL);
- Provide a review of recent adversarial approaches to IRL and IL;
- Introduce IRL methods which can surpass a demonstrator; and
- Review existing and potential applications of IRL and IL in quantitative finance.

The chapter is accompanied by a notebook comparing various IRL methods for the financial cliff walking problem. See Appendix “Python Notebooks” for further details.

2 Inverse Reinforcement Learning

The key idea of Russell (1998) is that the reward function should provide the most succinct representation of agents’ preferences while being transferable between both environments and agents. Before we turn to finance applications, imagine for a moment examples in everyday life—an adaptive smart home that learns from the habits of its occupants in scheduling different tasks such as pre-heating food, placing orders to buy other food, etc. In autonomous cars, the control system could learn from drivers’ preferences to set up an autonomous driving style that would be comfortable to a driver when taken for a ride as a passenger. In marketing applications, knowing preferences of customers or potential buyers, quantified as their utility functions, can inform marketing strategies tuned to their perceived preferences. In financial applications, knowing the utility of a counterparty may be useful in bilateral trading, e.g. over-the-counter (OTC) trades in derivatives or credit default swaps. Other financial applications of IRL, such as option pricing, will be discussed later in this chapter once we present the most popular IRL methods.

Just as reinforcement learning is rooted in dynamic programming, IRL has also its analog (or predecessor) in inverse optimal control (ICO). As with IRL, the objective of ICO is to learn the cost function. However, in the ICO setting, the dynamics and optimal policy are assumed to be known. Faithful to a data-driven approach of (direct) reinforcement learning, IRL does *not* assume that state dynamics or policy functions are known, and instead constructs an empirical distribution.²

Inverse reinforcement learning (IRL) therefore provides a useful extension (or inversion, hence justifying its name) of the (direct) RL paradigm. In the context of batch-mode learning used in the previous chapter, the setting of IRL is nearly identical to that of RL (see Eq. (10.58)), except that there is no information about the rewards:

$$\mathcal{F}_t^{(n)} = \left\{ \left(X_t^{(n)}, a_t^{(n)}, X_{t+1}^{(n)} \right) \right\}_{t=0}^{T-1}, \quad n = 1, \dots, N. \quad (11.1)$$

²Methods of ICO that assume that dynamics are known are sometimes referred to as model-based IRL in the machine learning literature.

The objective of IRL is typically two-fold: (i) find the rewards $R_t^{(n)}$ most consistent with observed states and action and (ii) find the optimal policy and action-value function (as in RL). One can distinguish between *on-policy* IRL and *off-policy* IRL. In the former case, we know that observed actions were *optimal* actions. In the latter case, observed actions may *not* necessarily follow an optimal policy and can be sub-optimal or noisy.

In general, IRL is a harder problem than RL. Indeed, not only must the optimal policy be found from data, which is the same task as in RL, but under the additional complexity that the rewards are unobserved. It appears that information about rewards is frequently missing in many potential real-world applications of RL/IRL. In particular, this is typically the case when RL methods are applied to study human behavior, see, e.g., Liu et al. (2013). IRL is also widely used in robotics as a useful alternative to direct RL methods via training robots by demonstrations (Kober et al. 2013).

It appears that IRL offers a very attractive, at least conceptually, approach for many financial applications that use rational agents involved in a sequential decision process, where no information about rewards received by an agent is available to a researcher. Some examples of such (semi-) rational agents would be retail or institutional investors, loan or mortgage borrowers, deposit or saving account holders, credit card holders, consumers of utilities such as cloud computing, mobile data, electricity, etc.

In the context of trading applications, such an IRL setting may arise when a trader seeks to learn a strategy of a counterparty. She observes the counterparty's actions in their bilateral trades, but not the counterparty's rewards. Clearly, if she reverse-engineered the most likely counterparty's rewards from the observed actions to find the counterparty's objective (strategy), she could use it to design her own strategy. This typifies an IRL problem.

Example 11.1 IRL for Financial Cliff Walking

Consider our financial cliff walking (FCW) example from Chap. 9 where we presented it as a toy problem for RL control—an over-simplified model of household finance. Now let us consider an IRL formulation of this problem where we would be given a set of trajectories sampled from some policy, and attempt to find both the rewards and the policy.

Clearly, as you will recall, the optimal policy for the FCW example is to deposit the minimum amount in the account at time $t = 0$, then take no further action until the very last step, at which point the account should be closed, with the reward of 10. However, sampling from this policy, possibly randomized by adding a random component, may miss the important penalty for breaching the bankruptcy level, and rather treat any examples of such events in the training data as occasional “sub-optimal” trajectories. As we will show in Sect. 9, the conventional IRL indeed misses the higher importance of not breaching the bankruptcy level than of achieving the final-step rewards.

2.1 RL Versus IRL

A very convenient concept for differentiating between the IRL and the direct RL problem is the occupancy measure $\rho_\pi(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ (see (Puterman 1994) and Exercise 9.6 in Chap. 9):

$$\rho_\pi(s, a|s_0) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s|\pi, s_0), \quad (11.2)$$

where $\Pr(s_t = s|\pi)$ is the probability density of the state $s = s_t$ at time t following policy π . The occupancy measure is also a function of the current state s_0 at time $t = 0$. The value function $V = V(s_0)$ of the current state can now be defined as an expectation of the reward function:

$$V(s_0) = \int \rho_\pi(s, a|s_0) r(s, a) ds da. \quad (11.3)$$

Recall from Exercise 9.1 that due to an invariance of the optimal policy under a common rescaling of all rewards $r(s, a) \rightarrow \alpha r(s, a)$ for some fixed $\alpha > 0$, the occupancy measure $\rho_\pi(s, a|s_0)$ can be interpreted as a normalized probability density of state-action pairs, even though the correct normalization is not explicitly enforced in the definition (11.2).

Here we arrive at the central difference between RL and IRL. In RL, we are given numerical values of an unknown reward function $r(s_t, a_t)$, where sampled trajectories $\tau = \{s_t, a_t\}_{t=0}^T$ of length T are obtained by using an unknown “expert” policy π_E (for batch-mode RL), or alternatively for on-line RL by sampling from the environment when executing a model policy $\pi_\theta(a|s)$. The problem of (direct) RL is to find an optimal policy π_\star , and hence an optimal measure ρ_\star that maximizes the expectation (11.3) given the sampled data in the form of triplets (s_t, a_t, r_t, s_{t+1}) . Because we observe numerical rewards, a value function can be directly estimated from data using Monte Carlo methods. Note that the optimal measure ρ_\star cannot be an arbitrary probability density function, but rather should satisfy time consistency constraints imposed by model dynamics, known as *Bellman flow constraints*.

Now compare this setting with IRL. In IRL, data consists of triplets (s_t, a_t, s_{t+1}) *without* rewards r_t . In other words, all we observe are trajectories—sequences of states and actions $\tau = \{s_t, a_t\}_{t=0}^T$. In terms of maximization of the value function as the expected value (11.3), the IRL setting amounts to providing a set of pairs $\{s_t, a_t\}_{t=0}^T$ that is assumed to be informative for a Monte Carlo based estimate of the expectation (11.3) of a (yet unknown) reward function $r(s_t, a_t)$.

Clearly, to build any model of the reward function given the observed trajectories, we should assume that trajectories demonstrated are sufficiently representative of true dynamics, and that the expert policy used in the recorded data is optimal or at least “sufficiently close” to an optimal policy. If either of these assumptions do not

hold, it is highly implausible that a “true” reward function can be recovered from such data.

On the other hand, if demonstrated trajectories are obtained from an expert, actions taken should be optimal from the viewpoint of Monte Carlo sampling of the expectation (11.3). A simple model that relates observed actions with rewards and encourages taking optimal actions could be a stochastic policy $\pi(a|s) \sim \exp(\beta(r(s, a) + F(s, a)))$, where $\beta > 0$ is a parameter (inverse temperature), $r(s, a)$ is the expected reward for a single step, and $F(s, a)$ is a function that incorporates information of future rewards into decision-making at time t .

As we saw in the previous chapter, Maximum Entropy RL produces exactly this type of policy, where $r(s, a) + F(s, a) = G_t^\pi(s_t, a_t) = \mathbb{E}^\pi[r(s_t, a_t, s_{t+1})] + \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) F_{t+1}^\pi(s_{t+1})$ is the G-function (the “soft” Q-value), see also Exercise 9.13 in Chap. 9. Due to exponential dependence of this policy on instantaneous rewards $r(s_t, a_t, s_{t+1})$ or equivalently on expected rewards $r(s_t, a_t) = \mathbb{E}^\pi[r(s_t, a_t, s_{t+1})]$, the optimal policy optimizes the expected total reward.

The idea of Maximum Entropy IRL (MaxEnt IRL) is to preserve the functional form of Boltzmann-like policies $\pi_\theta(a|s)$ produced by MaxEnt RL, where θ is a vector of model parameters. As they are explicit functions of states and actions, we can now use them differently, as probabilities of data in terms of observed values of states s_t and actions a_t . Parameters of the MaxEnt Boltzmann policy $\pi_\theta(a|s)$ can therefore be inferred using the standard maximum likelihood method. We shall return to MaxEnt IRL later in this chapter.

2.2 What Are the Criteria for Success in IRL?

In the absence of a “ground truth” expected reward function $r(s, a)$, what are the performance criteria for any IRL method that learns a parameterized reward policy $r_\theta(s, a) \in \mathcal{R}$, where \mathcal{R} is the space of all admissible reward functions?

Recall that the task of IRL is to learn *both* the reward function and optimal policy π and hence the state-action occupation measure ρ_π from the data. Therefore, once both functions are learned, we can use them both to compute the value function obtained with these inferred reward and policy functions. The quality of the IRL method would thus be determined by the value function obtained using these reward and policy functions.

We conclude that performance criteria for IRL involve solving a direct RL problem with a learned reward function. Moreover, we can make maximization of the expected reward an objective of IRL, such that each iteration over parameters θ specifying a parameterized expected reward function $r_\theta(s, a) \in \mathcal{R}$ will involve solving a direct RL problem with a current reward function. But such an IRL method could easily become very time-consuming and infeasible in practice for problems with large state-action space, where a direct RL problem would become computationally intensive.

Some methods of imitation learning or IRL avoid the need to solve a direct RL problem in an inner loop. For example, MaxEnt IRL methods that we mentioned above reduce IRL to a problem of inference in a graphical (more precisely, exponential) model. This changes the computational framework, but produces another computational burden, as it requires estimation of a normalization constant Z of a MaxEnt policy (also known as a partition function). Early versions of MaxEnt IRL for discrete state-action spaces computed such normalization constants using dynamic programming (Ziebart et al. 2008), see later in this chapter for more details. While more recent MaxEnt IRL approaches rely on different methods of estimation of the partition function Z (e.g., using importance sampling), this suggests that improving the computational efficiency of IRL methods by excluding RL from the internal loop is a hard problem. Such an approach is addressed by GAIL and related methods that will be presented later in this chapter.

2.3 Can a Truly Portable Reward Function Be Learned with IRL?

Recall the basic premise of reinforcement learning is that a reward function is the most compact form of expressing preferences of an agent. As an expected reward function $r(s_t, a_t)$ depends only on the current state and actions and does not depend on the dynamics, once it specified, it can be used with direct RL to find an optimal policy for any environment.

In the setting of IRL, we do not observe rewards but rather learn them from an observed behavior. As expected rewards are only functions of states and actions, and are independent of an environment, it appears appealing to try to estimate them by fitting a parameterized action policy π_θ to observations, and conjecture that the inferred reward would produce optimal policies in environments different from the one used for learning.

The main question, of course, is how realistic is such a conjecture? Note that this question is different (and in a sense harder) than the aforementioned problem of ill-posedness of IRL.

As an inverse problem, IRL is sensitive to the choice of a regularization method. A particular regularization may address the problem of incomplete data for learning a function, but it does not ensure that a reward learned with one environment would produce an optimal policy when an agent with a learned reward is deployed in a new environment that differs from an environment used for learning.

The concept of reward shaping suggested by Ng and Russell in 1999 (see Exercise 9.5 in Chap. 9) suggests that the problem of learning a portable (robust) reward function from demonstrations can be challenging. The main result of this analysis is that the optimal policy remains unchanged under the following transformation of the instantaneous reward function $r(s, a, s')$:

$$\tilde{r}(s, a, s') = r(s, a, s') + \gamma \Phi(s') - \Phi(s) \quad (11.4)$$

for an arbitrary function $\Phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. As we will see next, the reward shape invariance of reinforcement learning may indicate that the problem of learning robust (portable) rewards may be quite challenging and infeasible to solve in a model-independent way.

To see this, assume that we have two MDPs M and M' that have identical rewards and differ only in transition probabilities that we will denote as T and T' , respectively. A simple example would be deterministic dynamics $T(s, a) \rightarrow s'$. If a “true” reward function is of the form (11.4), it can be expressed as $r(s, a, s') + \gamma T\Phi(T(s, a)) - \Phi(s)$. If the new dynamics is such that $T'(s, a) \neq T(s, a)$, such reward will not be in the equivalence class of shape invariance for dynamics T' (Fu et al. 2019).

On the other hand, the same argument suggests an approach for constructing a reward function that could be transferred to other environments. To this end, we could simply *constrain* the inferred expected reward *not* to contain any additive contributions that would depend only on the state s_t . In other words, we can learn a reward function only up to an arbitrary additive function of the state. Due to the reward shape invariance of an optimal policy, this function is of no interest for finding the optimal policy, and thus can be set to zero for all practical purposes.

? Multiple Choice Question 1

Select all the following correct statements:

- a. The task of inverse reinforcement learning is to learn the dynamics from the observed behavior.
 - b. The task of inverse reinforcement learning is to find the worst, rather than the best policy for the agent.
 - c. The task of inverse reinforcement learning is to find both the reward function and the policy from observations of the states and actions of the agent.
-

3 Maximum Entropy Inverse Reinforcement Learning

In learning from demonstrations, it is important to understand which assumptions are made regarding actions performed by an agent. As both the policy and rewards are unknown in the setting of IRL, we have to make additional assumptions when solving this problem. A natural assumption would be to expect that the demonstrated actions are optimal or close to optimal. In other words, it means that the agent acts optimally or close to optimally.

If we assume that the agent follows a deterministic policy, then the above assumption implies that *every* action should be optimal. But this leaves little margin

for any errors resulting from model bias, noisy parameter estimations, etc. Under a deterministic policy model, a trajectory that contains a single sub-optimal action has an infinite negative log-likelihood—which exactly expresses the impossibility of such scenarios with deterministic policies.

A more forgiving approach is to assume that the agent followed a stochastic policy $\pi_\theta(a|s)$. Under such a policy, sub-optimal actions can be observed, though they are expected to be suppressed according to a model for $\pi_\theta(a|s)$. Once a parametric model for the stochastic policy is specified, its parameters can be estimated using standard methods such as maximum likelihood estimation.

In this section, we present a family of probabilistic IRL models with a particular exponential specification of a stochastic policy:

$$\pi_\theta(a|s) = \frac{1}{Z_\theta(s)} e^{\hat{r}_\theta(s,a)}, \quad Z_\theta(s) = \int e^{\hat{r}_\theta(s,a)}. \quad (11.5)$$

Here $\hat{r}_\theta(s, a)$ is some function that will be related to the reward and action-value functions of reinforcement learning. We have already seen stochastic policies with such exponential specification in Chap. 9 when we discussed “softmax in action” policies (see Eq. (9.38)), and in Chap. 10 when we introduced G-learning (see Eq. (10.114)).

While in the previous sections a stochastic policy with an exponential parameterization such as Eq. (11.5) was occasionally referred to as a softmax policy, in the setting of inverse reinforcement learning, it is often referred to as a *Boltzmann policy*, in recognition of its links to statistical mechanics and work of Ludwig Boltzmann in the nineteenth century (see the box below).

Methods leading to stochastic policies of the form (11.5) are generally based on maximization of entropy or minimization of the KL divergence in a space of parameterized action policies. The objective of this section is to provide an overview of such approaches.



The Boltzmann Distribution in Statistical Mechanics

In statistical mechanics, exponential distributions similar to (11.5) appear when considering closed systems with a fixed composition, such as molecular gases, that are in thermal equilibrium with their environment. The first formulation was suggested by Ludwig Boltzmann in 1868 in his work that developed a probabilistic approach to molecular gases in thermal equilibrium. The Boltzmann distribution characterizes states of a macroscopic system, such as a molecular gas, in terms of their energies E_i , where index i enumerates possible states. When such a system is at equilibrium with its

(continued)

environment that has temperature T , the Boltzmann distribution gives the probabilities of different energy states in the following form:

$$p_i = \frac{1}{Z} e^{-\frac{E_i}{k_B T}},$$

where k_B is a constant parameter called the Boltzmann constant, and Z is a normalization factor of the distribution, which is referred to as the *partition function* in statistical mechanics. The Boltzmann distribution can be obtained as a distribution which maximizes the entropy of the system

$$H = - \sum_i p_i \log p_i,$$

where the sum is taken over all energy states accessible to the system, subject to the constraint $\sum_i p_i E_i = \bar{E}$, where \bar{E} is some average mean energy. The Boltzmann distribution was extensively investigated and generalized beyond molecular gases to a general setting of systems at equilibrium by Josiah Willard Gibbs in 1902. It is therefore often referred to as the Boltzmann–Gibbs distribution in physics. In particular, Gibbs introduced the notion of a *statistical ensemble* as an idealization consisting of virtual copies of a system, such that each copy represents a possible state that the real system might be in. The physics-based notion of an ensemble corresponds to the notion of a probability space in mathematics. The Boltzmann distribution arises for the so-called *canonical ensemble* that is obtained for a system with a fixed number of particles at thermal equilibrium with its environment (called a heat bath in physics) at a fixed temperature T . The Boltzmann–Gibbs distribution serves as the foundation for the modern approach to equilibrium statistical mechanics (Landau and Lifshitz 1980).

3.1 Maximum Entropy Principle

The Maximum Entropy (MaxEnt) principle (Jaynes 1957) is a general and highly popular method for ill-posed inversion problems where a probability distribution should be learned from a finite set of integral constraints on this distribution. The main idea of the MaxEnt method for inference of distributions from data given constraints is that beyond matching such constraints, the inferred distribution

should be maximally non-informative, i.e. it should produce the highest possible uncertainty of a random variable described by this distribution.

As an amount of uncertainty in a distribution can be quantified by its entropy, the MaxEnt method amounts to finding the distribution that maximizes the entropy while matching all available integral constraints. The MaxEnt principle provides a practical implementation of Laplace's principle of insufficient reason, and has roots in statistical physics (Jaynes 1957).

Here we show the working of the MaxEnt principle using an example of learning the action policy in a simple single-step reinforcement setting. Such a setting is equivalent to removing time from the problem. This simplifies the setup considerably, because all data can now be assumed *i.i.d.*, and there is no need to consider future implications of a current action. The resulting time-independent version of the MaxEnt principle is a version originally proposed by Jaynes in 1957. To understand this approach, we shall elucidate it from a statistical mechanics perspective.

Let $\pi(a|s)$ be an action policy. Consider a one-step setting, with a single reward $r(s, a)$ to be received for different combinations of (s, a) . An optimal policy should maximize the value function $V^\pi(s) = \int r(s, a)\pi(a|s)da$. However, the value function is a linear functional of $\pi(a|s)$ and does not exhibit an optimal value of $\pi(a|s)$ per se. Assuming that the rewards $r(s, a)$ are known, a concave optimization problem is obtained if we add an entropy regularization and consider the following functional:

$$F^\pi(s) := V^\pi(s) + \frac{1}{\beta} H[\pi(a|s)] = \int \pi(a|s) \left[r(s, a) - \frac{1}{\beta} \log \pi(a|s) \right] da, \quad (11.6)$$

where $1/\beta$ is a regularization parameter. If we take the variational derivative of this expression with respect to $\pi(a|s)$ and set it to zero, we obtain the optimal action policy (see Exercise 11.1)

$$\pi(a|s) = \frac{1}{Z_\beta(s)} e^{\beta r(s, a)}, \quad Z_\beta(s) := \int e^{\beta r(s, a)} da. \quad (11.7)$$

Clearly, this expression assumes that the reward function $r(s, a)$ is such that the integral defining the normalization factor $Z_\beta(s)$ converges for all attainable values of s . Equation (11.7) has the same exponential form as Eq. (11.5) if we choose a parametric specification $\beta r(s, a) = r_\theta(s, a)$.

The expression (11.7) is obtained above as a solution of an entropy-regularized maximization problem. The same form can also be obtained in a different way, by maximizing the entropy of $\pi(a|s)$ conditional on matching a given average reward $\bar{r}(s)$. This is achieved by maximizing the following functional:

$$\tilde{F}^\pi(s) = - \int \pi(a|s) \log \pi(a|s) + \lambda \left(\int \pi(a|s) r(s, a) da - \bar{r}(s) \right), \quad (11.8)$$

where λ is a Lagrange multiplier. The optimal distribution is

$$\pi(a|s) = \frac{1}{Z_\lambda(s)} e^{\lambda r(s,a)}, \quad Z_\lambda(s) = \int e^{\lambda r(s,a)} da. \quad (11.9)$$

This has the same form as (11.7) with $\beta = 1/\lambda$. On the other hand, for problems involving integral constraints, the value of λ can be fixed in terms of the expected reward \bar{r} . To this end, we substitute the solution (11.9) into Eq. (11.8) and minimize the resulting expression with respect to λ to give:

$$\min_{\lambda} \log Z_\lambda - \lambda \bar{r}(s). \quad (11.10)$$

This produces

$$\frac{1}{Z_\lambda(s)} \int r(s, a) e^{\lambda r(s,a)} da = \bar{r}(s), \quad (11.11)$$

which is exactly the integral constraint on $\pi(a|s)$. The optimal value of λ is found by solving Eq. (11.11) or equivalently by numerical minimization of (11.10). Note that this produces a unique solution as the optimization problem (11.10) is convex (see Exercise 11.1).

► Links with Statistical Mechanics

As especially suggested by the second derivation, Eq. (11.7) or (11.9) can also be seen as a Boltzmann distribution of a statistical ensemble with energies $E(s, a) = -r(s, a)$ on a space of state-action pairs. In statistical mechanics, a distribution of states $x \in X$ in a space of state X with energies $E(x)$ for a canonical ensemble with a fixed average energy is given by the same Boltzmann form, albeit with a different functional form of the energy E . In statistical mechanics, parameter β has a specific form $\beta = 1/(k_B T)$, where k_B is the Boltzmann constant, and T is a temperature of the system. For this reason, parameter β in Eq. (11.7) is often referred to as the *inverse temperature*.

A direct generalization of the MaxEnt principle is given by the minimum cross-entropy (MCE) principle that replaces the absolute entropy with a KL divergence with some reference distribution $\pi_0(a|s)$. In this case, instead of (11.8), we consider the following KL-regularized value function:

$$F^\pi(s) = \int \pi(a|s) \left[r(s, a) - \frac{1}{\beta} \log \frac{\pi(a|s)}{\pi_0(a|s)} \right] da. \quad (11.12)$$

The optimal action policy for this case is

$$\pi(a|s) = \frac{1}{Z_\beta(s)} \pi_0(a|s) e^{\beta r(s,a)}, \quad Z_\beta(s) := \int \pi_0(a|s) e^{\beta r(s,a)} da. \quad (11.13)$$

The common feature of all methods based on MaxEnt or MCE principles is therefore the appearance of an exponential energy-based probability distribution.

For a simple single-step setting with reward $r(s, a)$, the MaxEnt optimal policy is exponential in $r(s, a)$. As we will see next, an extension of entropy-based analysis also produces an exponential specification for the action policy in a multi-step case.

3.2 Maximum Causal Entropy

In general, reinforcement learning or inverse reinforcement learning involves trajectories that extend over multiple steps. In the most general form, we have a series of states $\mathbf{S}^T = \mathbf{S}_{0:T}$ and a series of actions $\mathbf{A}^T = \mathbf{A}_{0:T}$ with some trajectory length $T > 1$. Sequences of states and actions in an MDP can be thought of as two interacting random processes $\mathbf{S}_{0:T}$ and $\mathbf{A}_{0:T}$. The problem of learning a policy can be viewed as a problem of inference of a distribution of actions $\mathbf{A}_{0:T}$ given a distribution of states $\mathbf{S}_{0:T}$.

Unlike MaxEnt inference problems for i.i.d. data, the time dependence of such problems requires some care. Indeed, a naive definition of a distribution of actions conditioned on states could involve conditional probabilities defined on the whole paths, such as $P[\mathbf{A}_{0:T} | \mathbf{S}_{0:T}]$. A problem with such a definition would be that it could violate causality, as conditioning on a whole path of states involves conditioning on the future. For Markov Decision Processes, actions at time t can depend only on the current state. If memory effects are important, they can be handled by using higher-order MDPs, or by switching to autoregressive models such as recurrent neural networks. Clearly, in both cases, to preserve causality, actions now (at time t) cannot depend on the future.

When each variable a_t is conditioned only on a portion $S_{0:t}$ of all variables $S_{0:T}$, the probability of \mathbf{A} *causally conditioned* on \mathbf{S} reads

$$P\left(\mathbf{A}^T \middle\| \mathbf{S}^T\right) = \prod_{t=0}^T P(A_t | S_{0:t}, \mathbf{A}_{0:t-1}). \quad (11.14)$$

Note that the standard definition of conditional probability would involve conditioning on the whole path $\mathbf{S}_{0:T}$ —which would violate causality. The causal conditional

probability (11.14) implies, in particular, that any joint distribution $P(\mathbf{A}^T, \mathbf{S}^T)$ can be factorized as $P(\mathbf{A}^T, \mathbf{S}^T) = P(\mathbf{A}^T || \mathbf{S}^T) P(\mathbf{S}^T || \mathbf{A}^{T-1})$.

The *causal entropy* (Kramer 1998) is defined as follows:

$$H(\mathbf{A}^T || \mathbf{S}^T) = \mathbb{E}_{\mathbf{A}, \mathbf{S}} \left[-\log P(\mathbf{A}^T || \mathbf{S}^T) \right] = \sum_{t=0}^T H(\mathbf{A}_t || \mathbf{S}_{0:t}, \mathbf{A}_{0:t}). \quad (11.15)$$

In this section, we assume that the dynamics are Markovian, so that $P(\mathbf{S}^T || \mathbf{A}^{T-1}) = \prod_t P(s_{t+1} | s_t, a_t)$. In addition, we assume a setting of an infinite-horizon MDP. For this case, we should use a discounted version of the causal entropy:

$$H(\mathbf{A}_{0:\infty} || \mathbf{S}_{0:\infty}) = \sum_{t=0}^T \gamma^t H(\mathbf{A}_t || \mathbf{S}_{0:t}, \mathbf{A}_{0:t}), \quad (11.16)$$

with a discount factor $\gamma \leq 1$.

The causal entropy (11.16) (or (11.15), for a finite-horizon case) presents a natural extension of an entropy of a conditional distribution to a dynamic setting where conditional information changes over time. This is precisely the case for learning policies for Markov Decision Processes. In this setting, states s_t of a system can be considered conditioning information, while actions a_t are the subject of learning.

For a first-order MDP, probabilities of actions at time t depend only on the state at time t , and the causal entropy of the action policy takes a simpler form that depends only on a policy distribution $\pi(a_t | s_t)$:

$$H(\mathbf{A}_{0:\infty} || \mathbf{S}_{0:\infty}) = \sum_{t=0}^{\infty} \gamma^t H(a_t || s_t) = -\mathbb{E}_{\mathbf{S}} \left[\sum_{t=0}^{\infty} \gamma^t \int \pi(a_t | s_t) \log \pi(a_t | s_t) da_t \right], \quad (11.17)$$

where the expectation is taken over all future values of s_t . Importantly, because the process is Markov, this expectation depends only on marginal distributions of s_t at $t = 0, 1, \dots$, but not on their joint distribution.

The causal entropy can be maximized under available constraints, providing an extension of the MaxEnt principle to dynamic processes. Let us assume that some feature functions $\mathcal{F}(\mathbf{S}, \mathbf{A})$ are observed in a demonstration with T step and that feature functions are additive in time, i.e. $\mathcal{F}(\mathbf{S}, \mathbf{A}) = \sum_t F(s_t, a_t)$. In particular, a total reward obtained from a trajectory $(\mathbf{S}^T, \mathbf{A}^T)$ is additive in time; therefore, it would fit such a choice. For additive feature functions, we have

$$\mathbb{E}_{\mathbf{A}, \mathbf{S}}^{\pi} [\mathcal{F}(\mathbf{S}, \mathbf{A})] = \mathbb{E}_{\mathbf{A}, \mathbf{S}}^{\pi} \left[\sum_{t=0}^{\infty} \gamma^t F(s_t, a_t) \right]. \quad (11.18)$$

Here $\mathbb{E}_{\mathbf{A}, \mathbf{S}}^\pi [\cdot]$ denotes expectation w.r.t. a distribution over future states and actions induced by the policy $\pi(a_t | s_t)$ and transition dynamics of the system expressed via conditional transition probabilities $P(s_{t+1} | s_t, a_t)$.

Suppose we seek a policy $\pi(a|s)$ which matches empirical feature expectations

$$\tilde{\mathbb{E}}_{emp} [\mathcal{F}(\mathbf{S}, \mathbf{A})] = \mathbb{E}_{emp} \left[\sum_{t=0}^{\infty} \gamma^t F(s_t, a_t) \right] = \frac{1}{T} \sum_{t=0}^{T-1} \gamma^t F(s_t, a_t). \quad (11.19)$$

Maximum Causal Entropy optimization can now be formulated as follows:

$$\begin{aligned} & \underset{\pi}{\operatorname{argmax}} H \left(\mathbf{A}^T \middle\| \mathbf{S}^T \right) \\ & \text{Subject to: } \mathbb{E}_{\mathbf{A}, \mathbf{S}}^\pi [\mathcal{F}(\mathbf{S}, \mathbf{A})] = \tilde{\mathbb{E}}_{\mathbf{A}, \mathbf{S}} [\mathcal{F}(\mathbf{S}, \mathbf{A})] \\ & \text{and } \sum_{a_t} \pi(a_t | s_t) = 1, \quad \pi(a_t | s_t) \geq 0, \quad \forall s_t. \end{aligned} \quad (11.20)$$

Here $\tilde{\mathbb{E}}_{\mathbf{A}, \mathbf{S}} [\cdot]$ stands for the empirical feature expectation as in Eq.(11.19). In contrast to a single-step MaxEnt problem, constraints now refer to feature expectations collected over whole paths rather than single steps. Causality is not explicitly enforced in (11.21); however, a causally conditioned policy in an MDP factorizes as $\prod_{t=0}^{\infty} \pi_t(a_t | s_t)$. Therefore, using the factors $\pi(a_t | s_t)$ as decision variables, the policy π is forced to be causally conditioned.

Equivalently, we can swap the objective and the constraints, and consider the following dual problem:

$$\begin{aligned} & \underset{\pi}{\operatorname{argmax}} \mathbb{E}_{\mathbf{A}, \mathbf{S}}^\pi [\mathcal{F}(\mathbf{S}, \mathbf{A})] - \tilde{\mathbb{E}}_{\mathbf{A}, \mathbf{S}} [\mathcal{F}(\mathbf{S}, \mathbf{A})] \\ & \text{Subject to: } H \left(\mathbf{A}^T \middle\| \mathbf{S}^T \right) = \bar{H} \\ & \text{and } \sum_{a_t} \pi(a_t | s_t) = 1, \quad \pi(a_t | s_t) \geq 0, \quad \forall s_t, \end{aligned} \quad (11.21)$$

where \bar{H} is some value of the entropy that is fixed throughout the optimization. Unlike the previous formulation that is non-concave and involves an infinite number of constraints, the dual formulation *is* concave, and involves only one constraint on the entropy (in addition to normalization constraints).

The dual form (11.21) of the Max-Causal Entropy method can be used for both direct RL and IRL. We first consider applications of this approach to direct reinforcement learning problems where rewards are observed.

? Multiple Choice Question 2

Select all the following correct statements:

- a. The Maximum Causal Entropy method provides an extension of the maximum entropy method for sequential decision-making inference which preserves causality relations between actions and future states.
 - b. The dual form of Maximum Causality Entropy produces multiple solutions as it amounts to a non-concave optimization.
 - c. A causality-conditioned policy with the Maximum Causality Entropy method is ensured by adding causality constraints.
 - d. A causality-conditioned policy with the Maximum Causality Entropy method is ensured by the MDP factorization of the process.
-

3.3 G-Learning and Soft Q-Learning

To apply the Max-Causal Entropy model (11.21) to reinforcement learning with observed rewards, we take expected instantaneous rewards $r(s_t, a_t)$ as features, i.e. set $F(s_t, a_t) = r(s_t, a_t)$. Furthermore, for direct reinforcement learning, the second term in the objective function (11.21) depends only on the empirical measure but not the policy $\pi(a|s)$, and therefore it can be dropped from the optimization objective function. Finally, we extend the Max-Causal Entropy method by switching to a KL divergence with some reference policy $\pi_0(a|s)$ instead of using the entropy of $\pi(a|s)$ as a regularization. The latter case can always be recovered from the former one by choosing a uniform reference density $\pi_0(a|s)$.

The Kullback–Leibler (KL) divergence of $\pi(\cdot|s_t)$ and $\pi_0(\cdot|s_t)$ is

$$KL[\pi || \pi_0](s_t) := \sum_{\mathbf{a}_t} \pi(\mathbf{a}_t | s_t) \log \frac{\pi(\mathbf{a}_t | s_t)}{\pi_0(\mathbf{a}_t | s_t)} = \mathbb{E}_{\pi} [g^{\pi}(\mathbf{s}, \mathbf{a}) | s_t], \quad (11.22)$$

where

$$g^{\pi}(\mathbf{s}, \mathbf{a}) = \log \frac{\pi(\mathbf{a}_t | s_t)}{\pi_0(\mathbf{a}_t | s_t)} \quad (11.23)$$

is the one-step *information cost* of a learned policy $\pi(\mathbf{a}_t | s_t)$ relative to a reference policy $\pi_0(\mathbf{a}_t | s_t)$.

The problem of policy optimization expressed by Eqs. (11.21), generalized here by using the KL divergence (11.22) instead of the causal entropy, can now be formulated as a problem of maximization of the following functional:

$$F_t^\pi(\mathbf{s}_t) = \sum_{t'=t}^T \gamma^{t'-t} \mathbb{E} \left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \frac{1}{\beta} g^\pi(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right]. \quad (11.24)$$

where $1/\beta$ is a Lagrange multiplier. Note that we dropped the second term in the objective function (11.21) in this expression. The reason is that this term depends only on the reward function but not on the policy. Therefore, it can be omitted for the problem of direct reinforcement learning. Note however that it should be kept for IRL as we will discuss later.

The expression (11.24) is a value function of a problem with a modified KL-regularized reward $r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \frac{1}{\beta} g^\pi(\mathbf{s}_{t'}, \mathbf{a}_{t'})$ that is sometimes referred to as the *free energy* function. Note that β in Eq. (11.24) serves as the “inverse-temperature” parameter that controls a tradeoff between reward optimization and proximity to the reference policy. The free energy $F_t^\pi(\mathbf{s}_t)$ is the entropy-regularized value function, where the amount of regularization can be calibrated to the level of noise in the data.

The optimization problem (11.24) is exactly the one we studied in Chap. 10 where we presented G-learning. We recall a self-consistent set of equations that need to be solved jointly in G-learning:

$$F_t^\pi(\mathbf{s}_t) = \frac{1}{\beta} \log Z_t = \frac{1}{\beta} \log \sum_{\mathbf{a}_t} \pi_0(\mathbf{a}_t | \mathbf{s}_t) e^{\beta G_t^\pi(\mathbf{s}_t, \mathbf{a}_t)}. \quad (11.25)$$

$$\pi(\mathbf{a}_t | \mathbf{s}_t) = \pi_0(\mathbf{a}_t | \mathbf{s}_t) e^{\beta(G_t^\pi(\mathbf{s}_t, \mathbf{a}_t) - F_t^\pi(\mathbf{s}_t))}. \quad (11.26)$$

$$G_t^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t+1, \mathbf{a}} [F_{t+1}^\pi(\mathbf{s}_{t+1}) | \mathbf{s}_t, \mathbf{a}_t]. \quad (11.27)$$

Here the G-function $G_t^\pi(\mathbf{s}_t, \mathbf{a}_t)$ is a KL-regularized action-value function.

Equations (11.25, 11.26, 11.27) constitute a system of equations that should be solved self-consistently for $\pi(\mathbf{a}_t | \mathbf{y}_t)$, $G_t^\pi(\mathbf{y}_t, \mathbf{a}_t)$, and $F_t^\pi(\mathbf{y}_t)$ (Fox et al. 2015). For a finite-horizon problem of length T , the system can be solved by backward recursion for $t = T - 1, \dots, 0$, using appropriate terminal conditions at $t = T$.

If we substitute the augmented free energy (11.25) into Eq. (11.27), we obtain

$$G_t^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}_t, \mathbf{a}_t) + \frac{\gamma}{\beta} \mathbb{E}_{t+1, \mathbf{a}} \left[\log \sum_{\mathbf{a}_{t+1}} \pi_0(\mathbf{a}_{t+1} | \mathbf{s}_{t+1}) e^{\beta G_{t+1}^\pi(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})} \right]. \quad (11.28)$$

This equation is a soft relaxation of the Bellman optimality equation for the action-value function (Fox et al. 2015). The “inverse-temperature” parameter β in Eq. (11.28) determines the strength of entropy regularization. In particular, if we take $\beta \rightarrow \infty$, we recover the original Bellman optimality equation for the Q-function. Because the last term in (11.28) approximates the $\max(\cdot)$ function when β is large but finite, Eq. (11.28) is known, for a special case of a uniform reference density π_0 , as “soft Q-learning.”

Note that we could also bypass the G -function altogether, and proceed with the Bellman optimality equation for the free energy F-function (11.24). In this case, we have a pair of equations for $F_t^\pi(\mathbf{s}_t)$ and $\pi(\mathbf{a}_t|\mathbf{s}_t)$:

$$\begin{aligned} F_t^\pi(\mathbf{s}_t) &= \mathbb{E}_{\mathbf{a}} \left[r(\mathbf{s}_t, \mathbf{a}_t) - \frac{1}{\beta} g^\pi(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}} [F_{t+1}^\pi(\mathbf{s}_{t+1})] \right] \\ \pi(\mathbf{a}_t|\mathbf{s}_t) &= \frac{1}{Z_t} \pi_0(\mathbf{a}_t|\mathbf{s}_t) e^{r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}} [F_{t+1}^\pi(\mathbf{s}_{t+1})]}. \end{aligned} \quad (11.29)$$

Equation (11.29) shows that one-step rewards $r(\mathbf{s}_t, \mathbf{a}_t)$ do *not* form an alternative specification of single-step action probabilities $\pi(\mathbf{a}_t|\mathbf{s}_t)$. Rather, a specification of the sum $r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}} [F_{t+1}^\pi(\mathbf{s}_{t+1})]$ is required (Ortega et al. 2015). However, in a special case when dynamics are *linear* and rewards $r(\mathbf{s}_t, \mathbf{a}_t)$ are *quadratic*, the term $\mathbb{E}_{t, \mathbf{a}} [F_{t+1}^\pi(\mathbf{s}_{t+1})]$ has the same parametric form as the time- t reward $r(\mathbf{s}_t, \mathbf{a}_t)$; therefore, addition of this term amounts to a “renormalization” of parameters of the one-step reward function (see Exercise 11.3). When the objective is to learn a policy, such renormalized parameters can be directly learned from data, bypassing the need to separate them into the current reward and an expected future-reward.

We see that the G-learning (or equivalently the Max-Causal Entropy) optimal policy (11.26) for an MDP formulation is still given by the familiar Boltzmann form as in (11.7), but with a different energy function $\hat{\mathcal{F}}(s_t, a_t)$, which is now given by the G-function. Unlike the previous single-step case, in a multi-period setting this function is not available in a closed form, but rather defined recursively by the self-consistent G-learning equations (11.25, 11.26, 11.27).

3.4 Maximum Entropy IRL

The G-learning (Max-Causal Entropy) framework can be used for both direct and inverse reinforcement learning. Here we apply it for the task of IRL.

In this section, we assume a time-homogeneous MDP with an infinite time horizon. We also assume that a time-stationary action-value function $G(\mathbf{s}_t, \mathbf{a}_t)$ is specified as a parametric model (e.g., a neural network) with parameters θ , so we write it as $G_\theta(\mathbf{s}_t, \mathbf{a}_t)$. The objective of IRL inference is to learn parameters θ from data.

We start with the G-learning equations (11.25) expressed in terms of the policy function $\pi(\mathbf{a}_t|\mathbf{s}_t)$ and the parameterized G-function $G_\theta(\mathbf{s}_t, \mathbf{a}_t)$:

$$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \frac{1}{Z_\theta(\mathbf{s}_t)} \pi_0(\mathbf{a}_t|\mathbf{s}_t) e^{\beta G_\theta(\mathbf{s}_t, \mathbf{a}_t)}, \quad Z_\theta(\mathbf{s}_t) := \int \pi_0(\mathbf{a}_t|\mathbf{s}_t) e^{\beta G_\theta(\mathbf{s}_t, \mathbf{a}_t)} d\mathbf{a}_t, \quad (11.30)$$

where the G-function (a.k.a. a soft Q-function) satisfies a soft relation of the Bellman optimality equation

$$G_\theta(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}_t, \mathbf{a}_t) + \frac{\gamma}{\beta} \mathbb{E}_{t, \mathbf{a}} \left[\log \sum_{\mathbf{a}_{t+1}} \pi_0(\mathbf{a}_{t+1} | \mathbf{s}_{t+1}) e^{\beta G_\theta(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})} \right]. \quad (11.31)$$

Under MaxEnt IRL, the stochastic action policy (11.30) is used as a probabilistic model of observed data made of pairs $(\mathbf{s}_t, \mathbf{a}_t)$. A loss function in terms of parameters θ can therefore be obtained by applying the conventional maximum likelihood method to this model.

To gain more insight, let us start with the likelihood of a particular path τ :

$$\begin{aligned} P(\tau) &= p(\mathbf{s}_0) \prod_{t=0}^{T-1} \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) P(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \\ &= p(\mathbf{s}_0) \prod_{t=0}^{T-1} \frac{1}{Z_\theta(\mathbf{s}_t)} \pi_0(\mathbf{a}_t | \mathbf{s}_t) e^{\beta G_\theta(\mathbf{s}_t, \mathbf{a}_t)} P(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t). \end{aligned} \quad (11.32)$$

Now we take a negative logarithm of this expression to get the negative log-likelihood, where we can drop contributions from the initial state distribution $p(\mathbf{s}_0)$ and state transition probabilities $P(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$, as neither depends on the model parameters θ :

$$\mathcal{L}(\theta) = \sum_{t=0}^{T-1} (\log Z_\theta(\mathbf{s}_t) - \beta G_\theta(\mathbf{s}_t, \mathbf{a}_t)). \quad (11.33)$$

Minimization of this loss function with respect to parameters θ gives an optimal soft Q-function $G_\theta(\mathbf{s}_t, \mathbf{a}_t)$. Once this function is found, if needed or desired, we could use Eq. (11.31) in reverse to estimate one-step expected rewards $r(\mathbf{s}_t, \mathbf{a}_t)$.

Gradients of the loss function (11.33) can be computed as follows:

$$\begin{aligned} \frac{\partial \mathcal{L}(\theta)}{\partial \theta} &= \sum_{t=0}^{T-1} \left(\int \pi_\theta(\mathbf{a} | \mathbf{s}_t) \frac{\partial G_\theta(\mathbf{s}_t, \mathbf{a})}{\partial \theta} d\mathbf{a} - \frac{\partial G_\theta(\mathbf{s}_t, \mathbf{a}_t)}{\partial \theta} \right) \\ &= \langle \frac{\partial G_\theta(\mathbf{s}, \mathbf{a})}{\partial \theta} \rangle_{model} - \langle \frac{\partial G_\theta(\mathbf{s}, \mathbf{a})}{\partial \theta} \rangle_{data}. \end{aligned} \quad (11.34)$$

The second term in this expression can be computed directly from the data for any given value of θ , and thus does not pose any issues. The problem is with the first term in (11.34) that gives the gradient of the log-partition function in Eq. (11.33). This term involves an integral over all possible actions at each step of the trajectory, computed with the probability density $\pi_\theta(\mathbf{a} | \mathbf{s}_t)$. For a discrete-action MDP, the integral becomes a finite sum that can be computed directly, but for continuous and possibly high-dimensional action spaces, an accurate calculation of this integral for a fixed value of θ might be time-consuming. Given that this

integral should be evaluated multiple times during optimization of parameters θ , unless evaluated analytically, the computational burden of this step can be very high or even prohibitive.

Leaving these computational issues aside for a moment, the intermediate conclusion is that the task of IRL can be solved using the maximum likelihood method for the action policy produced by G-learning, and without explicitly solving the Bellman optimality equation. The solution approach used here is to model the whole soft action-value function $G_\theta(\mathbf{s}_t, \mathbf{a}_t)$ using a flexible function approximation method such as a neural network. As $G_\theta(\mathbf{s}_t, \mathbf{a}_t)$ defines the action policy $\pi_\theta(\mathbf{a}|\mathbf{s}_t)$, by making inference of the policy, we can directly learn the soft action-value function.

We note, however, that such apparent relief from the need to solve the (soft) Bellman optimality equation at each internal step of the IRL task has a flip side. In the form presented above, the MaxEnt IRL approach is identical to behavioral cloning with the G-function $G_\theta(\mathbf{s}_t, \mathbf{a}_t)$ that is fitted to data in the form of pairs $(\mathbf{s}_t, \mathbf{a}_t)$. This is different from TD methods such as Q-learning or its “soft” versions that consider triplets of transitions (s_t, a_t, s_{t+1}) , and thus capture dynamics of the system without estimating them explicitly. Therefore, all problems with behavioral cloning mentioned above in this chapter will also arise here. In particular, a soft value function using only pairs $(\mathbf{s}_t, \mathbf{a}_t)$ and maximum likelihood estimation could produce a G-function that would be compatible with the data, and yet produce implausible single-step reward functions when Eq.(11.31) is used at the last step with the estimated G-function.

To preclude such potential problems arising, instead of using a parametric model for the G-function, we could directly specify a parametric single-step reward function $r_\theta(\mathbf{s}_t, \mathbf{a}_t)$. For example, with linear architectures, a reward function is linear in a set of K pre-specified features $\Psi_k(\mathbf{s}_t, \mathbf{a}_t)$:

$$r_\theta(\mathbf{s}_t, \mathbf{a}_t) = \sum_{k=1}^K \theta_k \Psi_k(\mathbf{s}_t, \mathbf{a}_t). \quad (11.35)$$

Alternatively, a reward function can be non-linear in parameters θ , and could be defined using, e.g., a neural network or a Gaussian process.

> IRL with Bounded Rewards

As we discussed in Chap. 9, reinforcement learning requires that rewards should be bounded from above by some value r_{max} . On the other hand, it also has certain invariances with respect to transformations of the reward function, namely the policy remains unchanged under affine transformations

(continued)

of the reward function $r(s, a) \rightarrow ar(s, a) + b$, where $a > 0$ and b are fixed parameters. We can use this invariance in order to fix the highest possible reward to be zero: $r_{max} = 0$ without any loss of generality. Let us assume the following functional form of the reward function:

$$r(s, a) = \log D(s, a), \quad (11.36)$$

where $D(s, a)$ is another function of the state and action. Assume that the domain of function $D(s, a)$ is a unit interval, i.e. $0 \leq D(s, a) \leq 1$. In this case, the reward is bounded from above by zero, as required: $-\infty < r(s, a) \leq 0$.

Now, because $0 \leq D(s, a) \leq 1$, we can interpret $D(s, a)$ as a probability of a binary classifier. If $D(s, a)$ is chosen to be the probability that the given action a in state s was generated by the expert, then according to Eq. (11.36), maximization of the reward corresponds to maximization of log-probability of the expert trajectory.

Let us use a simple logistic regression model for $D(s, a) = \sigma(\theta\Psi(s, a))$, where $\sigma(x)$ is the logistic function, θ is a vector of model parameters of size K , and $\Psi(s, a)$ is a vector of K basis functions. For this specification, we obtain the following parameterization of the reward function:

$$r(s, a) = -\log \left(1 + e^{-\theta\Psi(s, a)} \right). \quad (11.37)$$

As one can check, this reward function is concave in a if basis functions $\Psi_k(s, a)$ are linear in a while having an arbitrary dependence on s (see Exercise 11.2). Therefore, such a reward can be used as an alternative to a linear specification (11.35) for risk-averse RL and IRL. We will return to the reward specification (11.36) below when we discuss imitation learning.

Once the reward function is defined, the parametric dependence of the G-function is fixed by the soft Bellman equation (11.31). The latter will also define gradients of the G-function that enter Eq. (11.34). The gradients can be estimated using samples from the true data-generating distribution π_E and the model distribution π_θ .

Clearly, solving the IRL problem in this way would make estimated rewards $r_\theta(\mathbf{s}_t, \mathbf{a}_t)$ more consistent with dynamics than in the previous version that directly works with a parameterized G-function. However, this forces the IRL algorithm to solve the direct RL problem of finding the optimal soft action-value function $G_\theta(\mathbf{s}_t, \mathbf{a}_t)$ at each step of optimization over parameters θ . Given that solving the direct RL problem even once might be quite time-consuming, especially in high-dimensional continuous action spaces, this can render the computational cost

of directly inferring one-step rewards very high and impractical for real-world applications.

Another and computationally more feasible option is to define the BC-like loss function (11.33) to be more consistent with the dynamics. We introduce a *regularization* that depends on observed triplet transitions (s_t, a_t, s_{t+1}) . One simple idea in this direction is to add a regularization term equal to the squared Bellman error, i.e. the squared difference between the left- and right-hand sides of Eq. (11.31), where the one-step reward is set to be a fixed number, rather than a function of state and action. Such an approach was applied in robotics where it was called SQIL (Soft Q Imitation Learning) (Reddy et al. 2019).

We will return to the topic of regularization in IRL in the next section, where we will consider IRL in the context of imitation learning. In passing, we shall consider other computational aspects of the IRL problem that persist with or without regularization.

3.5 Estimating the Partition Function

After parameters θ are optimized, Eq. (11.31) can be used in order to estimate one-step expected rewards $r(s_t, a_t)$. In practice, computing the gradients of the resulting loss function involves integration over a (multi-dimensional) action space. This produces the main computational bottleneck of the MaxEnt IRL method. Note that the same bottleneck arises in direct reinforcement learning with G-learning Eqs. (11.25) that also involves computing integrals over the action space.

One commonly used approach to numerically computing integrals involving probability distributions is importance sampling. If $\hat{\mu}(a_t|s_t)$ is a sampling distribution, then the integral appearing in the gradient (11.34) can be evaluated as follows:

$$\int \pi_\theta(a_t|s_t) \frac{\partial G_\theta(s_t, a_t)}{\partial \theta} da_t = \int \hat{\mu}(a_t|s_t) \frac{\pi_\theta(a_t|s_t)}{\hat{\mu}(a_t|s_t)} \frac{\partial G_\theta(s_t, a_t)}{\partial \theta} da_t, \quad (11.38)$$

which replaces integration with respect to the original distribution with the sampling distribution $\hat{\mu}(a_t|s_t)$, with the idea that this distribution might be easier to sample from than the original probability density. When this distribution is used for sampling, the gradients $\partial G_\theta/\partial \theta$ are multiplied by the likelihood ratios $\pi_\theta/\hat{\mu}$.

Importance sampling becomes more accurate when the sampling distribution $\hat{\mu}(a_t|s_t)$ is close to the optimal action policy $\pi_\theta(a_t|s_t)$. This observation could be used to produce an *adaptive* sampling distribution $\hat{\mu}(a_t|s_t)$. For example, we could envision a computational scheme with updates of the G-function according to the gradients

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta} = \sum_{t=0}^{T-1} \left(\int \hat{\mu}(a_t|s_t) \frac{\pi_\theta(a_t|s_t)}{\hat{\mu}(a_t|s_t)} \frac{\partial G_\theta(s_t, a)}{\partial \theta} da - \frac{\partial G_\theta(s_t, a_t)}{\partial \theta} \right), \quad (11.39)$$

which would be intermittent with updates of the sampling distribution $\hat{\mu}(\mathbf{a}_t | \mathbf{s}_t)$ that would depend on values of θ from a previous iteration. Such methods are known in robotics as “guided cost learning” (Finn et al. 2016). We will discuss a related method in Sect. 5 where we consider alternative approaches to learning from demonstrations. Before turning to such advanced methods, we would like to present a tractable formulation of MaxEnt IRL where the partition function can be computed exactly so that approximations are not needed. Without too much loss of generality, we will present such a formulation in the context of a problem of inference of customer preferences and price sensitivity. Such a problem can also be viewed as a special case of a consumer credit problem. Similar examples in consumer credit might include prepaid household utility payment plans where the consumer prepays their utilities and is penalized for overage and lines of credit in payment processing and ATM services. Other examples include consumer loans and mortgages where different loan products are offered to the consumer, with varying interest rates and late payment penalties, and the user chooses when to make principal payments.

? Multiple Choice Question 3

Select all the following correct statements:

- a. Maximum Entropy IRL provides a solution of the self-consistent system of G-learning without access to the reward function that fits observable sequences of states and actions.
- b. “Soft Q-learning” is a method of relaxation of the Bellman optimality equation for the action-value function that is obtained from G-learning by adopting a uniform reference action policy.
- c. Maximum Entropy IRL assumes that all demonstrations are strictly optimal.
- d. Taking the limit $\beta \rightarrow \infty$ in Maximum Entropy IRL is equivalent to assuming that all demonstrations are strictly optimal.

4 Example: MaxEnt IRL for Inference of Customer Preferences

The previous section presented a general formulation of the MaxEnt IRL approach. While this approach can be formulated for both discrete and continuous state-action spaces, for the latter case, computing the partition function is often the main computational burden in practical applications.

These computational challenges should not overshadow the conceptual simplicity of the MaxEnt IRL approach. In this section we present a particularly simple version of this method which can be derived using quadratic rewards and Gaussian policies. We will present this formulation in the context of a problem of utmost interest in marketing, which is the problem of learning preferences and price sensitivities of customers of a recurrent utility service.

We will also use this simple example to provide the reader with some intuition on the amount of data needed to apply IRL. As we will show below, caution should be exercised when applying IRL to real-world noisy data. In particular, using simulated examples, we will show how the observational noise, inevitable in any finite-sample data, can masquerade itself as an apparent heterogeneity of agents.

4.1 IRL and the Problem of Customer Choice

Understanding customer choices, demand, and preferences, with customers being consumers or firms, is a central tenet in the marketing literature. One important class of such problems is dynamic consumer demand for recurrent utility-like plans and services such as cloud computing plans, internet data plans, utility plans (e.g., electricity, gas, phone), etc. Consumer actions in this settings extend over a period of time, such as the term of a contract or a period between regular payments for a plan, and can therefore be considered a multi-step decision-making problem.

If customers are modeled as utility-maximizing rational agents, the problem is well suited for methods of inverse optimal control or inverse reinforcement learning. In the marketing literature, the inverse optimal control approach to learning the customer utility is often referred to as *structural models*, see, e.g., Marschinski et al. (2007). This approach has the advantage over purely statistical regression based models in its ability to discern true consumer choices and demand preferences from effects induced by particular marketing campaigns. This enables the promotion of new products and offers, whose attractiveness to consumers could then be assessed based on the learned consumer utility.

Structural models view forward-looking consumers as rational agents maximizing their streams of expected utilities of consumption over a planning horizon rather than their one-step utility. Structural models typically specify a model for a consumer utility, and then estimate such a model using methods of dynamic programming and stochastic optimal control.

Using the language of reinforcement learning, structural models require methods of dynamic programming or approximate dynamic programming using deterministic policies. As we mentioned earlier in this chapter, using deterministic policies to infer agents' utilities may be problematic if the demonstrated behavior is sub-optimal. A deterministic policy which assumes that each step should be *strictly* optimal, will assign a zero probability to any path that is not strictly optimal. This would rule out any data where the demonstrated behavior is expected to deviate in *any* way from a strictly optimal behavior. Needless to say, available data is almost always sub-optimal to a various extent.

To relax the assumption of strict optimality for all demonstrations, structural models usually add a random component to the one-step customer utility, which is sometimes referred to as "user shocks." An example of such an approach can be found in Xu et al. (2015) who applied it to infer reward (utility) functions of consumers of mobile data plans. While this enables sub-optimal trajectories,

this approach requires optimization of the reward parameters using Monte Carlo simulation, where unobserved and simulated “user shocks” are added in the parameter estimation procedure.

Instead of pursuing such an approach, MaxEnt IRL offers an alternative and more computationally efficient way to manage possible sub-optimality in data by using stochastic policies instead of deterministic policies. This approach provides some degree of tolerance to certain occasional, non-excessive, deviations from a strictly optimal behavior, which are described as rare fluctuations according to the model with optimal parameters.

We will now present a simple parametric specification of the MaxEnt IRL method that we introduced in this chapter. As we will show, it leads to a very lightweight computational method, in comparison to Monte Carlo based methods for structural models.

4.2 Customer Utility Function

More formally, consider a customer that purchased a single-service plan with the monthly price F , initial quota q_0 , and price p to be paid for the unit of consumption upon breaching the monthly quota on the plan.³ We specify a single-step utility (reward) function of a customer at time $t = 0, 1, \dots, T - 1$ (where T is a length of a payment period, e.g., a month) as follows:

$$r(a_t, q_t, d_t) = \mu a_t - \frac{1}{2} \beta a_t^2 + \gamma a_t d_t - \eta p(a_t - q_t)_+ + \kappa q_t \mathbb{1}_{a_t=0}. \quad (11.40)$$

Here $a_t \geq 0$ is the daily consumption on day t , $q_t \geq 0$ is the remaining allowance at the start of day t , and d_t is the number of remaining days until the end of the billing cycle, and we use a short notation $x_+ = \max(x, 0)$ for any x . The fourth term in Eq. (11.40) is proportional to the payment $p(a_t - q_t)_+$ made by the customer once the monthly quota q_0 is exhausted. Parameter η gives the price sensitivity of the customer, while parameters μ, β, γ specify the dependence of the user reward on the state-action variables q_t, d_t, a_t . Finally, the last term $\sim \kappa q_t \mathbb{1}_{a_t=0}$ gives the reward received upon zero consumption $a_t = 0$ at time t (here $\mathbb{1}_{a_t=0}$ is an indicator function that is equal to one if $a_t = 0$, and is zero otherwise). Model calibration amounts to estimation of parameters $\eta, \mu, \beta, \gamma, \kappa$ given the history of the user’s consumption.

Note that the reward (11.40) can be equivalently written as an expansion over $K = 5$ basis functions:

³For plans that do not allow breaching the quota q_0 , the present formalism still applies by setting the price p to infinity.

$$r(a_t, q_t, d_t) = \boldsymbol{\Theta}^T \Phi(a_t, q_t, d_t) = \sum_{k=0}^{K-1} \theta_k \Phi_k(a_t, q_t, d_t), \quad (11.41)$$

where

$$\begin{aligned}\theta_0 &= \mu \langle a_t \rangle, \quad \theta_1 = -\frac{1}{2} \beta \langle a_t^2 \rangle, \quad \theta_2 = \gamma \langle a_t d_t \rangle, \\ \theta_3 &= -\eta p \langle (a_t - q_t)_+ \rangle, \quad \theta_4 = \kappa \langle q_t \mathbb{1}_{a_t=0} \rangle\end{aligned}$$

(here $\langle X \rangle$ stands for the empirical mean of X), and the following set of basis functions $\{\Phi_k\}_{k=0}^{K-1}$ is used:

$$\begin{aligned}\Phi_0(a_t, q_t, d_t) &= a_t / \langle a_t \rangle, \\ \Phi_1(a_t, q_t, d_t) &= a_t^2 / \langle a_t^2 \rangle, \\ \Phi_2(a_t, q_t, d_t) &= a_t d_t / \langle a_t d_t \rangle, \\ \Phi_3(a_t, q_t, d_t) &= (a_t - q_t)_+ / \langle (a_t - q_t)_+ \rangle \\ \Phi_4(a_t, q_t, d_t) &= q_t \mathbb{1}_{a_t=0} / \langle q_t \mathbb{1}_{a_t=0} \rangle.\end{aligned} \quad (11.42)$$

As we explained above, structural models attempt to reconcile deterministic policies and a possible sub-optimal behavior by adding random “user shocks” to the user utility. For example, such “user shocks” can be added to parameter μ . A drawback of such an approach is that for model estimation, it requires Monte Carlo simulation of user shock paths.

This can be compared with MaxEnt IRL. Because MaxEnt IRL is a probabilistic approach that assigns probabilities to observed paths, it does *not* require introducing a random shock to the utility function in order to reconcile the model with a possible sub-optimal behavior. Therefore, MaxEnt IRL does *not* need Monte Carlo simulation to estimate parameters of the user utility, and instead can use standard maximum likelihood estimation (MLE). For the reward defined in Eq. (11.40), MLE amounts to a convex optimization with 5 variables, which can be performed efficiently using the standard off-the-shelf convex optimization software. Moreover, our specification (11.40) can be easily generalized by adding more basis functions while keeping the rest of the methodology intact.

4.3 Maximum Entropy IRL for Customer Utility

We use an extension of the MaxEnt IRL called Relative Entropy IRL (Boularias et al. 2011) which replaces the uniform distribution in the MaxEnt method by a non-uniform benchmark (or “prior”) distribution $\pi_0(a_t | q_t, d_t)$. This produces the exponential single-step transition probability:

$$\begin{aligned} P(q_{t+1} = q_t - a_t, a_t | q_t, d_t) &:= \pi(a_t | q_t, d_t) \\ &= \frac{\pi_0(a_t | q_t, d_t)}{Z_\theta(q_t, d_t)} \exp(r(a_t, q_t, d_t)) = \frac{\pi_0(a_t | q_t, d_t)}{Z_\theta(q_t, d_t)} \exp\left(\Theta^T \Phi(a_t, q_t, d_t)\right), \end{aligned} \quad (11.43)$$

where $Z_\theta(q_t, d_t)$ is a state-dependent normalization factor

$$Z_\theta(q_t, d_t) = \int \pi_0(a_t | q_t, d_t) \exp\left(\Theta^T \Phi(a_t, q_t, d_t)\right) da_t. \quad (11.44)$$

We note that most applications of MaxEnt IRL use multi-step trajectories as prime objects, and define the partition function Z_θ on the space of *trajectories*. While the first applications of MaxEnt IRL calculated Z_θ exactly for small discrete state-action spaces as in (Ziebart et al. 2008, 2013), for large or continuous state-action spaces we resort to approximate dynamic programming, or other approximation methods. For example, the Relative Entropy IRL approach of (Boularias et al. 2011) uses importance sampling from a reference (“background”) policy distribution to calculate Z_θ . It is this calculation that poses the main computational bottleneck for applications of MaxEnt/RelEnt IRL methods for large or continuous state-action spaces.

With a simple piecewise-quadratic reward such as Eq. (11.40), we can proceed differently: we define state-dependent normalization factors $Z_\theta(q_t, d_t)$ for *each* time step. Because we trade a path-dependent “global” partition function Z_θ for a local state-dependent factor $Z_\theta(q_t, d_t)$, we do not need to rely on exact or approximate dynamic programming to calculate this factor. This is similar to the approach of Boularias et al. (2011) (as it also relies on the Relative Entropy minimization), but in our case both the reference distribution $\pi_0(a_t | q_t, d_t)$ and normalization factor $Z_\theta(q_t, d_t)$ are defined on a single time step, and calculation of $Z_\theta(q_t, d_t)$ amounts to computing the integral (11.44). As we show below, this integral can be calculated analytically with a properly chosen distribution $\pi_0(a_t | q_t, d_t)$.

We shall use a mixture of discrete and continuous distribution for the reference (“prior”) action distribution $\pi_0(a_t | q_t, d_t)$:

$$\pi_0(a_t | q_t, d_t) = \bar{v}_0 \delta(a_t) + (1 - \bar{v}_0) \tilde{\pi}_0(a_t | q_t, d_t) \mathbb{I}_{a_t > 0}, \quad (11.45)$$

where $\delta(x)$ stands for the Dirac delta-function, and $\mathbb{I}_{x>0} = 1$ if $x > 0$ and zero otherwise. The continuous component $\tilde{\pi}_0(a_t | q_t, d_t)$ is given by a spliced Gaussian distribution

$$\tilde{\pi}_0(a_t | q_t, d_t) = \begin{cases} (1 - \omega_0(q_t, d_t)) \phi_1 \left(a_t, \frac{\mu_0 + \gamma_0 d_t}{\beta_0}, \frac{1}{\beta_0} \right) & \text{if } 0 < a_t \leq q_t \\ \omega_0(q_t, d_t) \phi_2 \left(a_t, \frac{\mu_0 + \gamma_0 d_t - \eta_0 p}{\beta_0}, \frac{1}{\beta_0} \right) & \text{if } a_t \geq q_t \end{cases}, \quad (11.46)$$

where $\phi_1(a_t, \mu_1, \sigma_1^2)$ and $\phi_2(a_t, \mu_2, \sigma_2^2)$ are probability density functions of two truncated normal distributions defined separately for small and large daily consumption levels, $0 \leq a_t \leq q_t$ and $a_t \geq q_t$, respectively (in particular, they both

are separately normalized to one). The mixing parameter $0 \leq \omega_0(q_t, d_t) \leq 1$ is determined by the continuity condition at $a_t = q_t$:

$$(1 - \omega_0(q_t, d_t))\phi_1\left(q_t, \frac{\mu_0 + \gamma_0 d_t}{\beta_0}, \frac{1}{\beta_0}\right) = \omega_0(q_t, d_t)\phi_2\left(q_t, \frac{\mu_0 + \gamma_0 d_t - \eta_0 p}{\beta_0}, \frac{1}{\beta_0}\right). \quad (11.47)$$

As this matching condition may involve large values of q_t where the normal distribution would be exponentially small, in practice it is better to use it by taking logarithms of both sides:

$$\omega_0(q_t, d_t) = \frac{1}{1 + \exp\left\{\log \phi_2\left(q_t, \frac{\mu_0 + \gamma_0 d_t - \eta_0 p}{\beta_0}, \frac{1}{\beta_0}\right) - \log \phi_1\left(q_t, \frac{\mu_0 + \gamma_0 d_t}{\beta_0}, \frac{1}{\beta_0}\right)\right\}}. \quad (11.48)$$

The prior mixing-spliced distribution (11.45), albeit represented in terms of simple distributions, leads to potentially quite complex dynamics that make intuitive sense and appear largely consistent with observed patterns of consumption. In particular, note that Eq. (11.46) indicates that large fluctuations $a_t > q_t$ are centered around a smaller mean value $\frac{\mu - \gamma d_t - \eta p}{\beta}$ than the mean value $\frac{\mu - \gamma d_t}{\beta}$ of smaller fluctuations $0 < a_t \leq q_t$. Both a reduction of the mean upon breaching the remaining allowance barrier and a decrease of the mean of each component with time appear quite intuitive in the current context. As will be shown below, a posterior distribution $\pi(a_t | q_t, d_t)$ inherits these properties while also further enriching the potential complexity of dynamics.⁴

The advantage of using the mixed-spliced distribution (11.45) as a reference distribution $\pi_0(a_t | q_t, d_t)$ is that the state-dependent normalization constant $Z_\theta(q_t, d_t)$ can be evaluated exactly with this choice:

$$Z_\theta(q_t, d_t) = \bar{v}_0 e^{\kappa q_t} + (1 - \bar{v}_0) (I_1(\theta, q_t, d_t) + I_2(\theta, q_t, d_t)), \quad (11.49)$$

where

$$\begin{aligned} I_1(\theta, q_t, d_t) &= (1 - \omega_0(q_t, d_t)) \sqrt{\frac{\beta_0}{\beta_0 + \beta}} \exp\left\{\frac{(\mu_0 + \mu + (\gamma_0 + \gamma)d_t)^2}{2(\beta_0 + \beta)} - \frac{(\mu_0 + \gamma_0 d_t)^2}{2\beta_0}\right\} \\ &\times \frac{N\left(-\frac{\mu_0 + \mu + (\gamma_0 + \gamma)d_t - (\beta_0 + \beta)q_t}{\sqrt{\beta_0 + \beta}}\right) - N\left(-\frac{\mu_0 + \mu + (\gamma_0 + \gamma)d_t}{\sqrt{\beta_0 + \beta}}\right)}{N\left(-\frac{\mu_0 + \gamma_0 d_t - \beta_0 q_t}{\sqrt{\beta_0}}\right) - N\left(-\frac{\mu_0 + \gamma_0 d_t}{\sqrt{\beta_0}}\right)} \\ I_2(\theta, q_t, d_t) &= \omega_0(q_t, d_t) \sqrt{\frac{\beta_0}{\beta_0 + \beta}} \exp\left\{\frac{(\mu_0 + \mu - (\eta_0 + \eta)p + (\gamma_0 + \gamma)d_t)^2}{2(\beta_0 + \beta)}\right\} \end{aligned} \quad (11.50)$$

⁴In particular, it promotes a static mixing coefficient v_0 to a state- and time-dependent variable $v_t = v(q_t, d_t)$.

$$-\frac{(\mu_0 - \eta_0 p + \gamma_0 d_t)^2}{2\beta_0} + \eta p q_t \Big\} \times \frac{1 - N\left(-\frac{\mu_0 + \mu - (\eta_0 + \eta)p + (\gamma_0 + \gamma)d_t - (\beta_0 + \beta)q_t}{\sqrt{\beta_0 + \beta}}\right)}{1 - N\left(-\frac{\mu_0 - \eta_0 p + \gamma_0 d_t - \beta_0 q_t}{\sqrt{\beta_0}}\right)},$$

where $N(x)$ is the cumulative normal probability distribution.

Probabilities of T -steps paths $\tau_i = \{a_t^i, q_t^i, d_t^i\}_{t=0}^T$ (where i enumerates different user-paths) are obtained as products of single-step probabilities:

$$P(\tau_i) = \prod_{(a_t, q_t, d_t) \in \tau_i} \frac{\pi_0(a_t | q_t, d_t)}{Z_\theta(q_t, d_t)} \exp\left(\Theta^T \Phi(a_t, q_t, d_t)\right) \sim \exp\left(\Theta^T \Phi^{(\tau_i)}(a_t, q_t, d_t)\right). \quad (11.51)$$

Here $\Phi^{(\tau_i)}(a_t, q_t, d_t) = \{\Phi_k^{(\tau_i)}(a_t, q_t, d_t)\}_{k=0}^{K-1}$ are cumulative feature counts along the observed path τ_i :

$$\Phi_k^{(\tau_i)}(a_t, q_t, d_t) = \sum_{(a_t, q_t, d_t) \in \tau_i} \Phi_k(a_t, q_t, d_t). \quad (11.52)$$

Therefore, the total path probability in our model is exponential in the total reward along a trajectory, as in the “classical” MaxEnt IRL approach (Ziebart et al. 2008), while the pre-exponential factor is computed differently as we operate with one-step, rather than path probabilities.

Parameters Θ defining the exponential path probability distribution (11.51) can be estimated by the standard maximum likelihood estimation (MLE) method. Assume we have N historically observed single-cycle consumption paths, and assume these path probabilities are independent.⁵ The total likelihood of observing these data is

$$L(\theta) = \prod_{i=1}^N \prod_{(a_t, q_t, d_t) \in \tau_i} \frac{\pi_0(a_t | q_t, d_t)}{Z_\theta(q_t, d_t)} \exp\left(\Theta^T \Phi(a_t, q_t, d_t)\right). \quad (11.53)$$

The negative log-likelihood is therefore, after omitting the term $\log \pi_0(a_t | q_t, d_t)$ that does not depend on Θ ,⁶ and rescaling by $1/N$,

$$-\frac{1}{N} \log L(\theta) = \frac{1}{N} \sum_{i=1}^N \left(\sum_{(q_t, d_t) \in \tau_i} \log Z_\theta(q_t, d_t) - \sum_{(a_t, q_t, d_t) \in \tau_i} \Theta^T \Phi(a_t, q_t, d_t) \right)$$

⁵A more complex case of co-dependencies between rewards for individual customers can be considered, but we will not pursue this approach here.

⁶Note that $Z_\theta(q_t, d_t)$ still depends on $\pi_0(a_t | q_t, d_t)$, see Eq. (11.44).

$$= \frac{1}{N} \sum_{i=1}^N \left(\sum_{(q_t, d_t) \in \tau_i} \log Z_\theta(q_t, d_t) - \Theta^T \Phi^{(\tau_i)}(a_t, q_t, d_t) \right). \quad (11.54)$$

Given an initial guess for the optimal parameter $\theta_k^{(0)}$, we can also consider a regularized version of the negative log-likelihood:

$$-\frac{1}{N} \log L(\theta) = \frac{1}{N} \sum_{i=1}^N \left(\sum_{(q_t, d_t) \in \tau_i} \log Z_\theta(q_t, d_t) - \Theta^T \Phi^{(\tau_i)}(a_t, q_t, d_t) \right) + \lambda \|\theta - \theta^{(0)}\|_q, \quad (11.55)$$

where λ is a regularization parameter, and $q = 1$ or $q = 2$ stands for the L_1 - and L_2 -norms, respectively. The regularization term can also be given a Bayesian interpretation as the contribution of a prior distribution on θ_k when the MLE estimation (11.53) is replaced by a Bayesian maximum a posteriori (MAP) estimation.

Using the well-known property that exponential models like (11.51) give rise to convex negative log-likelihood functions, our final objective function (11.55) is *convex* in parameters Θ (as can also be verified by a direct calculation), and therefore has a unique solution for any value of $\theta^{(0)}$ and λ . This ensures stability of the calibration procedure and a smooth evolution of estimated model parameters Θ between individual customers or between groups of customers.

The regularized negative log-likelihood function (11.55) can be minimized using a number of algorithms for convex optimization. If $\lambda = 0$ (i.e., no regularization is used), or $q = 2$, the objective function is differentiable, and gradient-based methods can be used to calibrate parameters θ_k . When $\lambda > 0$ and the L_1 -regularization is used, the objective function is non-differentiable at zero, which can be addressed by using the Orhant-Wise variant of the L-BFGS algorithm (Kalakrishnan et al. 2013).

4.4 How Much Data Is Needed? IRL and Observational Noise

IRL is typically harder than direct reinforcement learning because it is based on less data since the rewards are not observed. It is well known that training RL is often data intensive. For example, deep reinforcement learning often uses training datasets measured in millions of feature vector observations. For financial applications, even simple RL models with quadratic rewards that do not involve sophisticated function approximations may require tens of thousands of examples for sufficiently accurate results.

Given that IRL is typically harder than RL and that RL is typically so data intensive—for IRL to be successful, a natural first question we could ask is do we have enough data?

The answer to this question clearly depends on several factors. First, it depends on the model and its number of free parameters to tune. Second, it depends on the

purpose of using IRL. Consider, for example, the application of IRL to clustering of financial agents. The standard approach to clustering in general is to predetermine a set of features and then perform clustering with a metric defined on the Euclidean space of such features, such as a vector norm.

When the objects of clustering are not some abstract data points but rather financial agents, IRL offers an interesting and on many accounts very attractive alternative to this standard approach. The idea is to use the reward function that is learned by IRL for defining useful features for clustering. If clustering is performed in this way, the resulting clusters would differentiate between themselves by features that are correlated with an agents' rewards, and thus would be meaningful by construction.

This idea can be implemented in a very straightforward way. Assume that we have a set of N trajectories obtained with N different agents. Further assume that the reward function for each agent can be represented as an expansion over a set of K basis functions $\Psi_k(s_t, a_t)$:

$$r(s_t, a_t) = \sum_{k=1}^K \theta_k \Psi_k(s_t, a_t). \quad (11.56)$$

If IRL is performed separately for each agent, this results in a learned set of parameters $\{\theta_k\}$ that would in general be different for each agent. The learned parameters $\{\theta_k\}_{k=1}^K$ could then be used to define a metric, for example, the Euclidean distance between vectors between two agents i and j defined as $D_{ij} = \sum_{k=1}^K (\theta_k^{(i)} - \theta_k^{(j)})^2$.

Clearly, for this program to work, the differences between parameters θ should be statistically significant. Otherwise, heterogeneity of agents obtained in this way could be just an artifact of noise in data.

A prudent approach to discerning true heterogeneity in IRL data from spurious effects of sampling error is to rely on estimations on efficiency (e.g., rates of convergence) of different finite-sample estimators. In classical statistics, such rates can be obtained when the likelihood is analytically tractable. This is, for example, the case with maximum likelihood estimation using a squared loss function. In other cases, for example, in non-parametric models, the likelihood does not have a closed-form expression, and analytical results for convergence rates of pre-asymptotic (finite-sample) estimators are not available.

In such cases, a practical alternative to analytical formulae is to rely on Monte Carlo simulation. As parameters θ learned with IRL on a finite-size data can be viewed as statistical estimators, efficiency of such estimators and the impact of observational finite-sample noise on them can be explored by using them on data simulated from the same model. In other words, a model learned by IRL is assumed to be a true model of the world. If the model is generative, we can simulate from it using some fixed and plausible parameters, and produce an unlimited amount of data on demand. We can then plot histograms of predicted parameter values obtained for any given length T of observed sequences, and compare them with true values (those used to simulate the data), in order to assess finite-sample performance of IRL estimators.

The simple MaxEnt IRL framework, developed in this section, can help develop intuition on the amount of data needed to differentiate the true heterogeneity across agents from spurious effects of observational noise. As the model is generative and enables easy simulation, it can be used for assessing finite-sample performance using Monte Carlo simulation as we will describe next.

4.5 Counterfactual Simulations

After the model parameters Θ are estimated using the MLE method of Eq. (11.54) or (11.55), the model can be used for counterfactual simulations of total user rewards assuming that users adopt plans with different upfront premia F_j , prices p_j , and initial quota $q_j(0)$. To this end, note that given the daily consumption a_t and the previous values q_{t-1}, d_{t-1} , the next values are deterministic: $q_t = (q_{t-1} - a_t)_+$, $d_t = d_{t-1} - 1$. Therefore in our model, path probabilities are solely defined by action probabilities, and the probability density of different actions $a_t \geq 0$ at time t can be obtained from a one-step probability $P(\tau) \sim \exp(r(a_t, q_t, d_t))$. Using Eqs. (11.40) and (11.43), this gives

$$\pi(a_t|q_t, d_t) = \frac{\pi_0(a_t|q_t, d_t)}{Z_\theta(q_t, d_t)} \exp\left\{\mu a_t - \frac{1}{2}\beta a_t^2 + \gamma a_t d_t - \eta p(a_t - q_t)_+ + \kappa q_t \mathbb{1}_{a_t=0}\right\}. \quad (11.57)$$

Using the explicit form of a mixture discrete-continuous prior distribution $\pi_0(a_t|q_t, d_t)$ given by Eq. (11.45), we can express the “posterior” distribution $\pi(a_t|q_t, d_t)$ in the same form:

$$\pi(a_t|q_t, d_t) = v_t \delta(a_t) + (1 - v_t) \tilde{\pi}(a_t|q_t, d_t) \mathbb{I}_{a_t > 0}, \quad (11.58)$$

where the mixture weight becomes state- and time-dependent:

$$v_t = \frac{\bar{v}_0 \exp\{\kappa q_t\}}{Z_\theta(q_t, d_t)} = \frac{\bar{v}_0 \exp\{\kappa q_t\}}{\bar{v}_0 e^{\kappa q_t} + (1 - \bar{v}_0)(I_1(\theta, q_t, d_t) + I_2(\theta, q_t, d_t))}, \quad (11.59)$$

(here we used Eq. (11.49)), and the spliced Gaussian component is

$$\tilde{\pi}(a_t|q_t, d_t) = \begin{cases} (1 - \omega(\theta, q_t, d_t)) \phi_1\left(a_t, \frac{\mu_0 + \mu + (\gamma_0 + \gamma)d_t}{\beta_0 + \beta}, \frac{1}{\beta_0 + \beta}\right) & \text{if } 0 < a_t \leq q_t \\ \omega(\theta, q_t, d_t) \phi_2\left(a_t, \frac{\mu_0 + \mu - (\eta_0 + \eta)p + (\gamma_0 + \gamma)d_t}{\beta_0 + \beta}, \frac{1}{\beta_0 + \beta}\right) & \text{if } a_t \geq q_t \end{cases}, \quad (11.60)$$

where the weight $\omega(\theta, q_t, d_t)$ can be obtained using Eqs. (11.57) and (11.49). After some algebra, this produces the following formula:

$$\omega(\theta, q_t, d_t) = \frac{I_2(\theta, q_t, d_t)}{I_1(\theta, q_t, d_t) + I_2(\theta, q_t, d_t)} = \frac{1}{1 + \frac{I_1(\theta, q_t, d_t)}{I_2(\theta, q_t, d_t)}}, \quad (11.61)$$

where functions $I_1(\theta, q_t, d_t)$, $I_2(\theta, q_t, d_t)$ are defined above in Eqs. (11.50). The ratio $I_1(\theta, q_t, d_t)/I_2(\theta, q_t, d_t)$ can be equivalently represented in the following form:

$$\begin{aligned} & \frac{I_1(\theta, q_t, d_t)}{I_2(\theta, q_t, d_t)} \\ &= e^{-p(\eta_0+\eta)\left(q_t - \frac{\mu_0 + \mu + (\gamma_0 + \gamma)d_t}{\beta_0 + \beta}\right) - \frac{p^2(\eta_0 + \eta)^2}{2(\beta_0 + \beta)}} \frac{\int_0^{q_t} e^{-\frac{1}{2}(\beta_0 + \beta)\left(a_t - \frac{\mu_0 + \mu + (\gamma_0 + \gamma)d_t}{\beta_0 + \beta}\right)^2} da_t}{\int_{q_t}^{\infty} e^{-\frac{1}{2}(\beta_0 + \beta)\left(a_t - \frac{\mu_0 + \mu + (\gamma_0 + \gamma)d_t - (\eta_0 + \eta)p}{\beta_0 + \beta}\right)^2} da_t}. \end{aligned} \quad (11.62)$$

It can be checked by a direct calculation that Eq. (11.61) with the ratio $I_1(\theta, q_t, d_t)/I_2(\theta, q_t, d_t)$ given by Eq. (11.62) coincides with the formula for the weight that would be obtained from a continuity condition at $a_t = q_t$ if we started directly with Eq. (11.60). This would produce, similarly to Eq. (11.48),

$$\omega_0(q_t, d_t) = \frac{1}{1 + \exp \left\{ \log \frac{\phi_2\left(q_t, \frac{\mu_0 + \mu + (\gamma_0 + \gamma)d_t - (\eta_0 + \eta)p}{\beta_0 + \beta}, \frac{1}{\beta_0 + \beta}\right)}{\phi_1\left(q_t, \frac{\mu_0 + \mu + (\gamma_0 + \gamma)d_t}{\beta_0 + \beta}, \frac{1}{\beta_0 + \beta}\right)} \right\}}. \quad (11.63)$$

The fact that two expressions (11.61) and (11.63) coincide means that the “posterior” distribution $\pi(a_t|q_t, d_t)$ is continuous at $a_t = q_t$ as long as the “prior” distribution $\pi_0(a_t|q_t, d_t)$ is continuous there. Along with continuity at $a_t = q_t$, the optimal (or “posterior”) action distribution $\pi(a_t|q_t, d_t)$ has the same mixing discrete-spliced Gaussian structure as the reference (“prior”) distribution $\pi_0(a_t|q_t, d_t)$, while mixing weights, means, and variances of the component distributions are changed. Such a structure-preserving property of our model is similar in a sense to the structure-preservation property of conjugated priors in Bayesian analysis. Note that simulation from the spliced Gaussian distribution (11.60) is only slightly more involved than simulation from the standard Gaussian distribution. This involves first simulating a component of the spliced distribution, and then simulating a truncated normal random variable from this distribution. Different consumption paths are obtained by a repeated simulation from the mixing distribution (11.58), along with deterministic updates of the state variables q_t, d_t .

An example of simulated daily consumption using the mixed-spliced policy (11.58) is shown in Fig. 11.1, while the resulting trajectories for the remaining allowance are shown in Fig. 11.2 using model parameter values: $q_0 = 600$, $p = 0.55$, $\mu = 0.018$, $\beta = 0.00125$, $\gamma = 0.0005$, $\eta = 0.1666$, $\kappa = 0.0007$. In addition, we set $\mu_0 = \mu$, $\beta_0 = \beta$, $\gamma_0 = \gamma$, $\eta_0 = \eta$, $\kappa_0 = \kappa$, and $v_0 = 0.05$.

Note that consumption may vary quite substantially from one month to another (e.g., a customer can saturate their quota at about 80% of the time period, or can have a residual unused quota at the end of the month) purely due to the observational noise, even though the utility function remains the same.

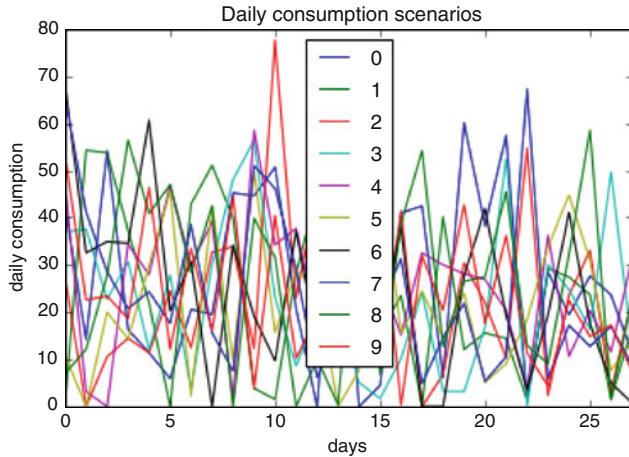


Fig. 11.1 Simulated daily consumption

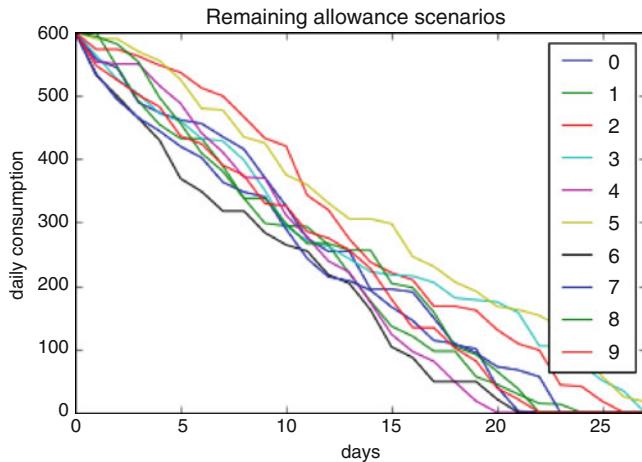


Fig. 11.2 Simulated remaining allowance

4.6 Finite-Sample Properties of MLE Estimators

While the maximum likelihood estimation (MLE) is known to provide asymptotically unbiased results for estimated model parameters, in practice we have to process data that has limited history at the granularity of individual customers. For example, the structural model of Xu et al. (2015) was trained on 9 months of data for 1000 customers. While the number of customers to be included for analysis can potentially be increased by collecting more data, collecting long *individual-level* consumption histories might be more difficult due to a number of factors such as, e.g., customer mobility.

In view of such potential limitations in the availability of long time series for service consumption, it is important to investigate finite-sample properties of the MLE estimators in the setting of our model. In particular, note that even if two customers have the *same* “true” model parameters, their finite-sample MLE estimates would in general be different for these customers.

Therefore, the ability of the model to differentiate between individual customers hinges upon the amount of bias and variance of its MLE estimator in realistically expected settings, with potentially limited amounts of data available for analysis. We note that Xu et al. (2015) reported a substantial heterogeneity of estimated model parameters for their dataset of 9 months of observations for 1000 users; however, they did not address the finite-sample properties of their estimators, thus ruling out the simplest interpretation of their results as being attributed to “observational noise” in their estimators. Such noise would be observed even for a perfectly homogeneous set of customers.

We have estimated the empirical distribution of MLE estimators for our model by repeatedly sampling N_m months of consumption history, which is performed N_p times, while keeping the model parameters fixed as per above. For each model parameter, we produce a histogram of its N_p estimated values.

The results are presented in Figs. 11.3, 11.4, 11.5, where we show the resulting histograms for $N_m = 10, 100$, and 1000 months of data, respectively, while keeping the number of experiments $N_p = 100$ for all graphs. Note that for all parameters except β , the standard deviation of the MLE estimate for $N_m = 10$ is nearly equal its mean. This implies that two users with 10 months of daily observations can hardly be differentiated by the model unless their implied parameters differ by a factor of two or more.

This might cast some doubts on a model-implied customer heterogeneity suggested in a similar setting by Xu et al. (2015), and suggests that some, if not all, of this heterogeneity can simply be explained by a *finite-sample noise* of a model estimation procedure, while all customers are actually undistinguishable from the model perspective.

On the other hand, one can see how both the bias and variance of MLE estimators decrease, as they should, with an increased span of the observation period from 10 user-months to 1000 user-months. These results suggest that in practice, the model should be calibrated using groups of customers with a similar consumption behavior. This can be implemented using widely available techniques for clustering time series.

4.7 Discussion

We have presented a tractable version of Maximum Entropy Inverse Reinforcement Learning (IRL) for a dynamic consumer demand estimation, which can be applied for designing appropriate marketing strategies for new products and services. The same approach can be applied, upon proper modifications to similar problems

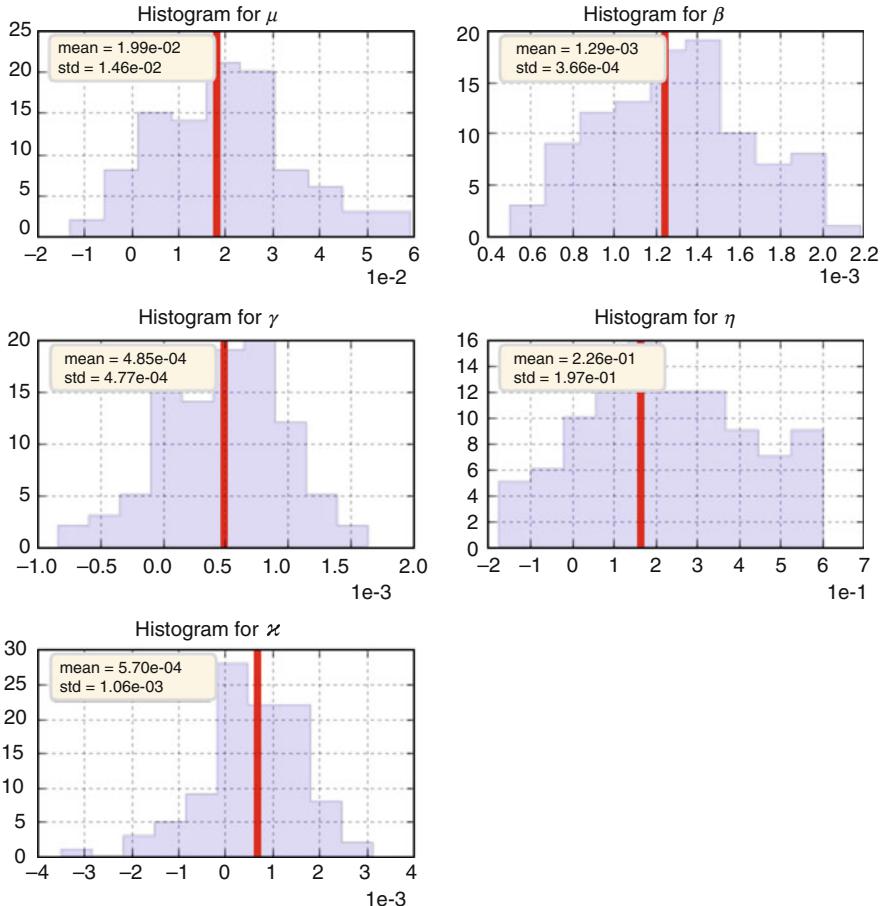


Fig. 11.3 Distributions of MLE estimators for $N_m = 10$ months of data

in marketing and pricing of recurrent utility-like services such as cloud plans, internet plans, electricity and gas plans, etc. The model enables easy simulations, which is helpful for conducting counterfactual experiments. On the IRL/machine learning side, unlike most of other versions of the Maximum Entropy IRL, our model does not have to solve a Bellman optimality equation even once. The model estimation in our approach amounts to convex optimization in a low-dimensional space, which can be solved using a standard off-the-shelf optimization software. This is much easier computationally than structural models that typically rely on Monte Carlo simulation for model parameter estimation. The availability of lightweight estimators in this model enables estimations of their finite-sample performance. In turn, this facilitates detection of true heterogeneity in data from spurious heterogeneity due to observational noise in the data.

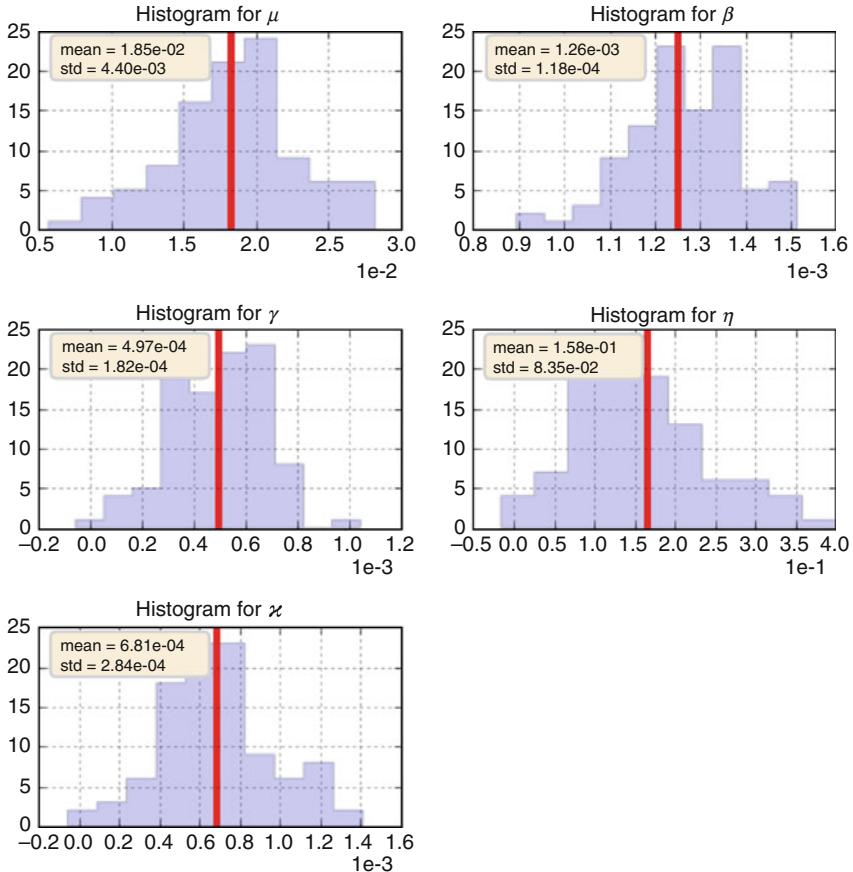


Fig. 11.4 Distributions of MLE estimators for $N_m = 100$ months of data

5 Adversarial Imitation Learning and IRL

5.1 Imitation Learning

Imitation learning is the second main class of models for learning from demonstrations. Unlike inverse reinforcement learning, imitation learning does not attempt to recover a reward function of an agent, but rather attempts to directly model the action policy given an observed behavior.

In imitation learning, the goal is to recover the expert policy rather than the reward function—it is clearly somewhat less ambitious than the goal of IRL which tries to find the reward function itself. While the latter is portable and can be used with a different environment, the former is not portable. Still, in many problems of practical interest, we do not need true portability. For example, in some trading

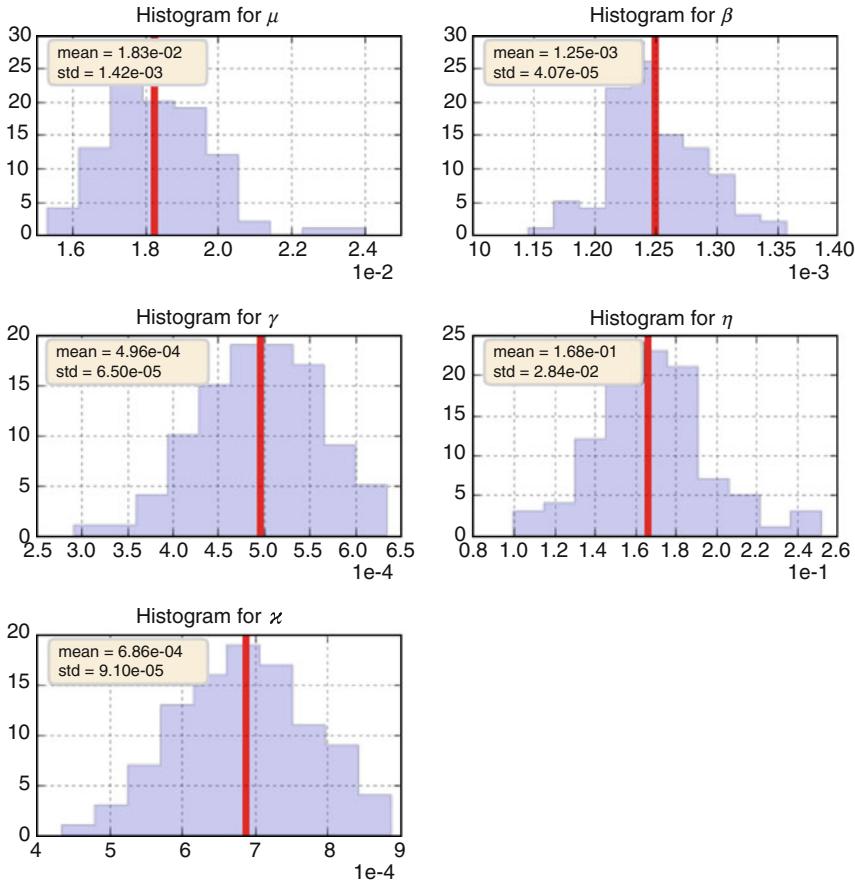


Fig. 11.5 Distributions of MLE estimators for $N_m = 1000$ months of data

applications—for short to intermediate time horizons (days or weeks)—we may reasonably posit approximate stationarity of the market dynamics, so that policies learned from historical data can be used to make trading decisions. In addition, as we saw in the previous section, a “true” reward function can only be learned up to some invariance transformations. These ambiguities of reward learning may make applications of IRL across different environments a delicate matter.

Imitation learning has approximately as long a history as IRL, but it substantially gained in popularity over the last few years among researchers who apply methods of RL and IRL in robotics and video games. For clarity of exposition, we note that behavioral cloning (BC) methods could also formally classify as imitation learning, as long as they do not explicitly introduce a reward function. Following the modern literature, in this book we shall not include BC as an imitation learning methods. Rather we shall build upon a stream of the literature that started with the generative adversarial imitation learning (GAIL) model of Ho and Ermon. Even

though GAIL does not recover the reward function, its generalizations, that we present later, do aim at recovering both the reward and policy function. Therefore, following conventions in the literature, we will sometimes refer to these methods as *adversarial IRL*.

5.2 GAIL: Generative Adversarial Imitation Learning

To introduce the idea of GAIL, recall the dual form of the Max-Causal Entropy method (11.21). In the previous section, we explained how Maximum Entropy IRL can be obtained using maximum likelihood inference of the model obtained by using Eq. (11.21) using the direct RL (more specifically, G-learning). With the latter approach, a generic feature $\mathcal{F}(\mathbf{S}, \mathbf{A})$ is replaced by a one-step expected reward function $r(s_t, a_t)$.

This line of reasoning thus proceeds in two steps: (i) assume a fixed and unknown reward function, and solve the direct RL problem with this function to produce an optimal policy and (ii) calibrate parameters of the reward function using the resulting optimal policy as a probabilistic model of observed actions with the conventional maximum likelihood method.

In practical IRL implementations, we usually rely on a parametric specification of a reward function. This implies that one should solve one direct RL problem per each iteration of the IRL optimization with respect to the reward function optimization.

Alternatively, we can consider a different IRL approach that is still based on the Max-Causal Entropy approach (11.21), but attempts to avoid solving the direct RL problem.

To this end, instead of solving a direct RL problem, assuming a fixed but unknown reward function and then maximizing the likelihood of data obtained with such model, we could replace a generic feature $\mathcal{F}(\mathbf{S}, \mathbf{A})$ by an unknown true reward function $r(s, a) \in \mathcal{R}$, and simultaneously optimize with respect to both the policy π and reward $r(s, a)$:

$$\max_{r(s,a) \in \mathcal{R}} \left(\max_{\pi} \mathbb{E}_{\rho_\pi} [r(s, a)] - \mathbb{E}_{\rho_E} [r(s, a)] \right),$$

where $\rho_\pi(s, a)$ and $\rho_E(s, a)$ are the joint state-action distributions induced by the learned policy π and the expert policy π_E (when the latter is available via samples), respectively, subject to a constraint on the causal entropy. Note the difference here with the previous formulation that skipped the second term as it is independent of π : When the true reward is unknown, this term should be kept.

To conform to a more common notation in the literature, we replace a reward function with a cost function (a negative reward) $c(s, a)$, and, respectively, replace maximization with respect to π by minimization. This produces the following IRL problem:

$$\max_{c(s,a) \in C} \min_{\pi} \mathbb{E}_{\rho_\pi} [c(s,a)] - \mathbb{E}_{\rho_E} [c(s,a)]$$

Subject to: $H^{causal}(\pi) = \bar{H}$, (11.64)

where $H^{causal}(\pi) = \mathbb{E}_{\rho_\pi} [\pi(a|s)]$ is the causal entropy, and \bar{H} is some pre-specified value of the causal entropy that is supposed to be fixed in optimization. We skipped here the normalization constraint for the policy π , though it is still assumed to hold for a solution of Eqs. (11.64).

> Maximum Entropy IRL and Apprenticeship Learning

The meaning of optimization in Eq. (11.64) is that we attempt to find an optimal policy π which produces at least as low total cost as an expert policy with the *unknown* cost function $c(s, a)$. We thus find the maximum over all such functions:

$$\max_{c(s,a) \in C} \min_{\pi} \mathbb{E}_{\rho_\pi} [c(s,a)] - \mathbb{E}_{\rho_E} [c(s,a)].$$

Clearly, such a problem is severely ill-posed in its present form, and the causal entropy regularization (the second line in (11.64)) selects a unique “best” solution among an infinite number of potential candidate solutions. While entropy regularization is a natural and very popular approach to regularization of IRL, it is not the only possible approach. In particular, the 2004 paper of Abbeel and Ng, “Apprenticeship Learning via Inverse Reinforcement Learning,” used a different regularization that amounts to maximizing the margin between the best policy and the second-best policy. The latter problem is similar to the problem of finding a maximum margin separation hyperplane in support vector machines (SVM) and, like the latter, the method of Abbeel and Ng relies on quadratic programming (QP).

Using a soft relaxation of the entropy constraint, this can be formulated as the following minimization problem:

$$\text{IRL}(\pi_E) = \max_{c(s,a) \in C} \min_{\pi} -H^{causal}(\pi) + \mathbb{E}_{\rho_\pi(s,a)} [c(s,a)] - \mathbb{E}_{\rho_E(s,a)} [c(s,a)]. (11.65)$$

At the minimum, the objective function returns the optimal cost function given the expert policy π_E . This optimization problem requires solving the direct RL problem

$$\text{RL}(c) = \min_{\pi} -H^{\text{causal}}(\pi) + \mathbb{E}_{\rho_{\pi}(s,a)} [c(s, a)] \quad (11.66)$$

in the inner loop of IRL optimization with respect to the true cost function.

? Where Is the Lagrange Multiplier?

Soft relaxation of Lagrange optimization problems with constraints typically amounts to replacing such constraints with soft penalties which are added directly to the objective function with a certain *fixed* weight. Unlike the true Lagrange multiplier, it is *not* subject to optimization, but is rather kept as a fixed parameter. In the limit when this parameter is taken to infinity, we recover a solution where the constraint is satisfied exactly. Still, note that we did not explicitly include any fixed regularization parameter in front of the entropy term in (11.65) (or equivalently, we set such a parameter to unity). Can you explain why this expression is still correct without losing any generality? *Hint:* Recall the previous discussion on ambiguities of recovering the true reward (or cost) function.

5.3 GAIL as an Art of Bypassing RL in IRL

GAIL focuses on problems where our final goal is to find an optimal policy (i.e., to solve the direct RL problem) when rewards are not observed, just as in the IRL setting. This can be stated as the problem of learning a policy π using RL with costs $\tilde{c}(s, a)$ that are recovered by IRL from the input expert policy π_E via samples collected with this policy. Formally, this can be written as a composite function

$$\text{RL} \circ \text{IRL}(\pi_E). \quad (11.67)$$

In this formula, optimization of the cost function is implied in the first layer of the composition, $\text{IRL}(\pi_E)$, according to Eq. (11.65). The standard approach to IRL is to assume a parametric function $c_\theta(s, a)$ and then try to learn its parameters. But this involves solving a direct RL problems multiple times in the inner loop of IRL, as we have just discussed. Our objective here is to find a framework that would avoid explicitly solving the direct RL problem when finding an optimal policy from demonstrations that do not include rewards.

An explicit RL problem could be avoided if we replaced a parametric specification with a *non-parametric* cost function defined on a space of *all* admissible costs functions. If optimization over such cost functions could be done analytically rather than numerically, the composite function (11.67) defining the combined RL-IRL objective could be implemented without solving a direct RL problem in the internal loop.

To see how this can be done, let us return to Eq.(11.65) and now add a regularization using a convex function $\psi(c)$:

$$\text{IRL}(\pi_E) = \max_{c(s,a) \in C} \min_{\pi} -\psi(c) - H^{\text{causal}}(\pi) + \mathbb{E}_{\rho_\pi(s,a)} [c(s,a)] - \mathbb{E}_{\rho_E(s,a)} [c(s,a)]. \quad (11.68)$$

? Why Is Regularization So Important for Imitation Learning?

We use this opportunity to make an important general remark about the role of regularization in machine learning. In some applications, regularization plays a somewhat secondary role relative to a main objective function. For example, ridge (L_2) regularization is frequently used for regression tasks, often providing a well-behaved solution of a least square optimization problem, often without significantly changing its functional form. In contrast, GAIL regularization is a critical key input to the model. As was found by Ho and Ermon, different choices of the regularization function $\psi(c)$ may produce many different practical implementations of imitation learning, including both the conventional behavioral cloning (BC) approaches and potentially new methods.

If we do not add *any* regularization, when optimized in a full functional space, Eq.(11.65) produces the result $\rho_\pi = \rho_E$, which is not useful in practice as it merely states that the agent should replicate the expert policy. When the available data does not cover the entire state-action space, this result is insufficient to produce a unique solution, calling for additional constraints. This is of course as expected since, without *any* constraints at all, the IRL problem is vastly ill-posed. To overcome this issue, we modify the objective (11.65) by adding a convex regularization function $\psi(c)$, so that it becomes (11.68).

Next, we note that the objective function in (11.68) is convex in π and concave in $c(s,a)$, due to convexity of the regularizer $\psi(c)$ (see Exercise 11.5). This means that the optimal solution $(\pi^*, c^*(s,a))$ is a saddle-point (a max-min solution) of

the objective (11.68). Instead of performing minimization first and maximization next, the order of these operations could be swapped for a saddle-point, producing an equivalent min–max solution, instead of a max–min solution (this is known as a strong duality in convex analysis). In other words, the two operations are exchangeable for the optimization problem (11.68). Using this, we write Eq. (11.68) as follows:

$$\begin{aligned} \text{RL} \circ \text{IRL}(\pi_E) &= \min_{\pi} -H^{\text{causal}}(\pi) + \max_{c(s,a) \in C} -\psi(c) \\ &\quad + \mathbb{E}_{\rho_{\pi}(s,a)} [c(s,a)] - \mathbb{E}_{\rho_E(s,a)} [c(s,a)]. \end{aligned} \quad (11.69)$$

Note that we wrote the result as $\text{RL} \circ \text{IRL}(\pi_E)$ rather than as $\text{IRL}(\pi_E)$ as in Eq. (11.65), bearing in mind that the meaning of a min–max solution of (11.69) is equivalent to the meaning of the composite function (11.67), where we solve the IRL problem in the inner layer, and then use the learned cost function to find the optimal policy in the outer (RL) layer.

Now we can formally integrate out the cost $c(s, a)$ in Eq. (11.69) as follows:

$$\begin{aligned} \text{RL} \circ \text{IRL}(\pi_E) &= \min_{\pi} -H^{\text{causal}}(\pi) \\ &\quad + \max_{c(s,a) \in C} \left(\sum_{s,a} (\rho_{\pi}(s, a) - \rho_E(s, a)) c(s, a) - \psi(c) \right) \\ &= \min_{\pi} -H^{\text{causal}}(\pi) + \psi^*(\rho_{\pi} - \rho_E), \end{aligned} \quad (11.70)$$

where $\psi^* : \mathbb{R}^{S \times \mathcal{A}} \rightarrow \bar{\mathcal{R}}$ is called the *convex conjugate* of the regularizer ψ :

$$\psi^*(x) = \sup_{y \in \mathbb{R}^{S \times \mathcal{A}}} \mathbf{x}^T \mathbf{y} - \psi(\mathbf{y}). \quad (11.71)$$

➤ Convex Conjugate

The convex conjugate function ψ^* is also known as a *Fenchel conjugate*. The transform (11.71) is also known as the Fenchel–Legendre transform. When the function $\psi(y)$ is convex and differentiable, as in our case, the convex conjugate coincides with the Legendre transform. If ψ^* is everywhere differentiable, then the pair ψ, ψ^* is dual in the sense that $\psi^{**} = \psi$ (see Exercise 11.4). Respectively, the inverse transform reads $\psi(x) = \sup_x \mathbf{y}^T \mathbf{x} - \psi^*(\mathbf{x})$.

The objective of the composite RL/IRL task (11.70) can therefore be formulated as finding the policy π that minimizes the difference between the occupancy measures $\rho_\pi(s, a)$ and $\rho_E(s, a)$, as measured by the convex conjugate ψ^* of the regularizer ψ , which is regularized by the causal entropy (Ho and Ermon 2016). Interestingly, Eq. (11.70) offers an interpretation of the cost function $c(s, a)$ as a (state-action dependent) Lagrange multiplier enforcing matching of occupancy measures $\rho_\pi(s, a)$ and $\rho_E(s, a)$.

To summarize so far, we have managed to eliminate the cost function optimization in (11.68), by switching to the convex conjugate ψ^* . However, in its most general form, with an arbitrary convex function $\psi(c)$, this is only a *formal* improvement, as it simply substitutes maximization over $c(s, a)$ in Eq. (11.68) by another maximization in the definition of the convex conjugate (11.71). However, as was shown by Ho and Ermon, certain choices of regularization $\psi(c)$ can lead to practically implementable schemes where the convex conjugate ψ^* is known in closed form. We will discuss such practical settings of GAIL next.

5.4 Practical Regularization in GAIL

Following Ho and Ermon (2016), we choose the Jensen–Shannon divergence $D_{JS}(\rho_\pi, \rho_E)$ for the convex conjugate regularization Ψ^* in Eq. (11.70), where

$$D_{JS}(\rho_\pi, \rho_E) = DK_{KL}\left(\rho_\pi \parallel \frac{1}{2}(\rho_\pi + \rho_E)\right) + DK_{KL}\left(\rho_E \parallel \frac{1}{2}(\rho_\pi + \rho_E)\right). \quad (11.72)$$

As the JS divergence is given by a linear combination of KL divergences both of which are non-negative by construction, it is also non-negative, and it vanishes only when $\rho_\pi = \rho_E$. Because it is symmetric with respect to the occupancy measures ρ_π and ρ_E , non-negative, and vanishes when $\rho_\pi = \rho_E$, the JS divergence is a valid metric defining the “distance” between measures ρ_π and ρ_E .

> F-Divergencies

The Jensen–Shannon (JS) divergence is not the only way to define a measure of similarity between two distributions. A more general definition which incorporates the JS distance as a special case is given by the notion of *f-divergences*. For any convex and lower-semi-continuous function $f : \mathbb{R}^+ \rightarrow$

(continued)

\mathbb{R} satisfying $f(1) = 0$, the f-divergence of two distributions P, Q with density functions p and q is defined as follows:

$$D_f(P \parallel Q) = \int q(x) f\left(\frac{p(x)}{q(x)}\right) dx. \quad (11.73)$$

In particular, if we take $f(x) = x \log x$, the f-divergence coincides with the KL divergence $D_{KL}(P \parallel Q)$, while $f(x) = -\log x$ gives rise to the reverse KL divergence $D_{KL}(Q \parallel P)$. A f-divergence of two distributions P, Q is called *symmetric* if we have $D_f(P \parallel Q) = D_f(Q \parallel P)$. In particular, the symmetric JS divergence is obtained with the choice $f(x) = x \log x - (x + 1) \log \frac{x+1}{2}$ (see Exercise 11.5). We will return to f-divergences in the next section.

The GAIL imitation learning algorithm is therefore defined as a problem of finding a policy π whose occupancy measure ρ_π minimizes the JS distance to the expert policy π_E , with regularization by the causal entropy of the policy π :

$$\text{RL} \circ \text{IRL}(\pi_E) = \min_{\pi} D_{JS}(\rho_\pi, \rho_E) - \lambda H^{causal}(\pi). \quad (11.74)$$

The JS distance regularization can also be interpreted as a loss function of a binary classification problem. Let $D(s, a)$ be the probability that the state-action pair (s, a) is generated by the expert policy π_E , and $1 - D(s, a)$ be the probability that this pair is generated by policy π . Such a classifier is referred to as a *discriminator* in GAIL, as it differentiates between policies π and π_E for state-action pairs generated from these policies. The JS divergence can then be written as follows:

$$\begin{aligned} \Psi_{GA}^*(\rho_\pi - \rho_E) &= D_{JS}(\rho_\pi, \rho_E) = \max_{D \in [0, 1]^{\mathcal{S} \times \mathcal{A}}} \mathbb{E}_{\pi_E} [\log D(s, a)] \\ &\quad + \mathbb{E}_\pi [\log (1 - D(s, a))], \end{aligned} \quad (11.75)$$

where maximization is performed over all admissible classifiers $D(s, a)$. As the reader can easily verify, such unconstrained maximization over a full functional space gives the result

$$D(s, a) = \frac{\rho_E(s, a)}{\rho_\pi(s, a) + \rho_E(s, a)}. \quad (11.76)$$

Substituting this expression back into Eq. (11.75) gives the JS divergence (11.72), plus a constant term (see Exercise 11.3).

➤ Trajectory Discriminator

Assume that we observed N state-action pairs (s_i, a_i) , where each pair is marked by a binary flag $b_i = \{0, 1\}$, where $b_i = 1$ if the pair (s_i, a_i) is generated by π_E , and $b_i = 0$ otherwise. The log-likelihood of such data is obtained as a log-product of Bernoulli likelihoods:

$$\begin{aligned}\mathcal{L} &= \frac{1}{N} \log \prod_{i=1}^N [D(s_i, a_i)]^{b_i} [1 - D(s_i, a_i)]^{1-b_i} \\ &= \frac{1}{N} \sum_{i=1} b_i \log D(s_i, a_i) + (1 - b_i) \log (1 - D(s_i, a_i)).\end{aligned}\quad (11.77)$$

When the number of observations is large, the ratios $\sum_i b_i/N$ and $\sum_i (1 - b_i)/N$ approach, respectively, the induced state-action densities ρ_E and ρ_π , so that in this limit we obtain

$$\mathcal{L} = \sum_{s,a} \rho_E(s, a) \log D(s, a) + \rho_\pi(s, a) \log (1 - D(s, a)), \quad (11.78)$$

which coincides with Eq. (11.75). Therefore, the log-likelihood of a binary trajectory classifier provides an empirical estimation of the JS divergence (11.72).

5.5 Adversarial Training in GAIL

Now we are ready to expand on the meaning of the word “adversarial” in the acronym, GAIL. To this end, let us recall the analysis of the optimization problem (11.74) with the binary log-loss (11.75). To compute the first term, we must take expectations with respect to the current policy π . For a continuous high-dimensional action space, this amounts to computing high-dimensional integrals in this space.

As we mentioned in Sect. 3.5, such integrals could be computed at a reasonable cost using importance sampling, if only we could find a convenient sampling distribution. As it is always the case with importance sampling, the *optimal* sampling distribution would be equal to the policy π , as can be seen from Eq. (11.75).

This suggests an iterative optimization procedure which keeps switching between two steps of maximization of the loss function (11.75), until convergence. In even steps, the loss function is computed using importance sampling for the first term, with a fixed sampling distribution. In odd steps, we update the sampling distribution to bring it closer to the optimal policy π and the expert policy π_E .

Generally, intractable expectations such as the first term in (11.74) can be computed using a parameterized transformation of random noise z_t (which can be, for example, white noise). More precisely, such a transformation can be implemented using a parameterized function $G_\theta(s_t, z_t)$, where z_t is a random noise with a simple distribution, e.g. a Gaussian noise. In particular, we can choose a linear transform

$$\pi_\theta(a_t|s_t) = G_\theta(s_t, z_t) = f_\theta(s_t) + \sigma_\theta(s_t)z_t, \quad (11.79)$$

where $f_\theta(s_t)$ and $\sigma_\theta(s_t)$ are two parameterized functions which can be implemented as, e.g., neural networks. If the noise z_t is Gaussian, $z_t \sim \mathcal{N}(0, 1)$, the output of the generator G_θ will be a Gaussian with state-dependent mean and variance given, respectively, by functions $f_\theta(s_t)$ and $\sigma_\theta(s_t)$. Sampling from such Gaussian distribution can be performed in numerically efficient ways. If needed or desired, one can use a non-Gaussian noise z_t , which would produce a non-Gaussian action policy.

Using the policy π_θ defined in Eq. (11.79), the derivative of the first term in (11.74) with respect to θ is (here $W(s, a) := \log(1 - D(s, a))$)

$$\nabla_\theta \mathbb{E}_{\pi_\theta} [W(s, a)] = \mathbb{E}_z [\nabla_a W(s, a) \nabla_\theta \pi_\theta(a|s)] \simeq \frac{1}{M} \sum_{i=1}^M \nabla_a W(s, a) \nabla_\theta \pi_\theta(a|s)|_{z=z_i}, \quad (11.80)$$

where z_i are samples from the distribution of noise z_t . With adversarial learning, Eq. (11.79) defines a *generator* whose objective is to produce samples from policy $\pi \sim G_\theta$. The optimization problem (11.74) is then interpreted as a min–max game between two players: the generator G_θ and the discriminator D implemented as a parameterized function $D_w(s, a)$ using, e.g., a neural network. The objective function of this game is obtained from Eqs. (11.74) and (11.75), where we substitute $D \rightarrow D_w$:

$$\text{RL} \circ \text{IRL}(\pi_E) = \min_{\pi \sim G_\theta} \max_{D_w \in [0, 1]} \mathbb{E}_{\pi_E} [\log D_w(s, a)] + \mathbb{E}_\pi [\log(1 - D_w(s, a))] - \lambda H^{causal}(\pi). \quad (11.81)$$

Optimization of this objective function (i.e., finding a saddle-point solution) is achieved by iterating between the actions of both players. In odd steps, Eq. (11.81) is maximized with respect to the discriminator D_w (i.e., with respect to its parameters) using a gradient descend update. In even steps, Eq. (11.81) is minimized with respect to parameters θ defining the action policy $\pi \sim G_\theta$ according to Eq. (11.79).

The min–max game (11.81) between the discriminator D_w and the policy generator G_θ employed by GAIL is a special case of generative adversarial training proposed by (Goodfellow et al. 2014) in their celebrated GANs (generative

adversarial networks) paper. In the original GAN framework for image recognition, a generator G_θ produces samples resembling real images. In GAIL, the generator produces samples from an action policy distribution.

Note that besides the regularization term, it is only the second term in (11.81) that depends on the policy $\pi \sim G_\theta$ produced by the generator G_θ . Recall that $D_w(s, a)$ is the probability that the discriminator identifies a given pair (s, a) as produced by expert policy π_E rather than by the model policy π_θ . Therefore, minimization of this term with respect to parameters of the policy generator $\pi \sim G_\theta$ attempts to maximally confuse the discriminator, and maximize the probability that the pair (s, a) produced by π is identified as being produced by the expert policy π_E .⁷ This suggests a simple metaphor of adversarial learning where the generator acts like a counterfeiter who produces fake money, while the discriminator acts as a regulator or intelligence agency attempting to eliminate counterfeit money (Goodfellow et al. 2014).

5.6 Other Adversarial Approaches*

The GAN paper by Goodfellow et al. (2014) has stimulated a substantial interest in the machine learning community, and a large number of useful and interesting extensions have followed. It was found that learning GANs turns out to be difficult in many cases because of instability of training. The instability is partially caused by the objective function of the original GAN, which suffers from vanishing gradients. This makes it difficult to train the generator. In this section, we briefly review various generalization and extensions of GAN.

5.7 *f*-Divergence Training*

As we already mentioned earlier in this section, the JS divergence is a special case of a symmetric f-divergence. For a general (either symmetric or asymmetric) case, the f-divergence is defined in Eq.(11.73) in Sect. 5.4, which we repeat here for convenience:

$$D_f(P||Q) = \int q(x) f\left(\frac{p(x)}{q(x)}\right) dx. \quad (11.82)$$

⁷The original GAN paper of Goodfellow et al. (2014) suggested that better performance is observed if instead of minimization of the second term in (11.81) which is equal to $\mathbb{E}_\pi [\log (1 - D_w(s, a))]$, one maximizes the expression $\mathbb{E}_\pi [\log D_w(s, a)]$ to the same effect of maximally confusing the discriminator.

Here $f : \mathbb{R}^+ \rightarrow \mathbb{R}$ is any convex and lower-semi-continuous function satisfying $f(1) = 0$. In particular, the symmetric JS divergence is obtained with the choice $f(x) = x \log x - (x + 1) \log \frac{x+1}{2}$ (see Problem 11.5).

As was shown by Nguyen et al. (2010) and Nowozin et al. (2016), one can use the inverse relation for the convex conjugate $f(x) = \sup_t xt - f^*(t)$ to get a variational representation of the f-divergence:

$$\begin{aligned} D_f(P || Q) &= \int q(x) \sup_{t \in \text{dom}_{f^*}} \left\{ t \frac{p(x)}{q(x)} - f^*(t) \right\} dx \\ &\geq \sup_{T \in \mathcal{T}} \left(\int p(x)T(x)dx - \int q(x)f^*(T(x)) \right) \\ &= \sup_{T \in \mathcal{T}} \mathbb{E}_{x \sim P}[T(x)] - \mathbb{E}_{x \sim Q}[f^*(T(x))]. \end{aligned} \quad (11.83)$$

Here $T : \mathcal{X} \rightarrow \mathbb{R}$ is an arbitrary function in a class \mathcal{T} . Equation (11.83) gives a lower bound on the f-divergence because of both Jensen's inequality and the fact that the \mathcal{T} may contain only a subset of all possible functions. In practical realizations, function T is implemented using a parameterized specification T_ω , where ω denotes a vector of tunable parameters, using, e.g., a neural network. This is used to construct an iterative optimization scheme for matching an implicit model distribution Q to a fixed distribution P using any f-divergence. The problem solved by this iterative method is the saddle-point min–max optimization problem (f-GAN)

$$\min_Q \max_{T_\omega} F(\theta, \omega) = \mathbb{E}_{x \sim P}[T_\omega] - \mathbb{E}_{x \sim Q}[f^*(T_\omega)]. \quad (11.84)$$

Practical parameterization methods of T_ω were considered in (Nowozin et al. 2016). We will return to Eq. (11.83) in the next section where we will show how it can be used to construct extensions of GAIL based on f-divergences that generalize the JS divergence approach of GAIL.

5.8 Wasserstein GAN*

As we mentioned above, notwithstanding enthusiasm in the machine learning community towards generative adversarial learning, practitioners found that training GAN or GAIL networks can be difficult due to instability of training. As we alluded to above, this instability is partially caused by the objective function of the original GAN.

To better understand the source of this issue, recall the main idea of GAN: Samples can be simulated from a model density P_θ by passing random noise (uniform, Gaussian, etc.) through a parametric function G_θ . In this case, P_θ would be an action policy for GAIL, or some other output, e.g. an image classifier for

other applications of GAN. If model parameters θ are trained using a loss function $\mathcal{L}(P_\theta, P_E)$, it is desirable to have a differentiable loss function for numerical efficiency. This implies that the mapping $\theta \rightarrow P_\theta$ should be differentiable (continuous). As in GAN and GAIL the loss function is given by a metric of a difference between distributions $\rho\pi$ and ρ_E , this means that the metric should be differentiable in θ .

For a continuous mapping $\theta \rightarrow P_\theta$, convergence in iteration of model parameter θ means convergence for the metric used for training (such as, e.g., the JS divergence). This assumes, however, that a density P_θ exists, but the problem is that such a density does *not* exist for some low-dimensional manifolds, as will be illustrated shortly.

As an alternative to using the JS divergence, Arjovsky et al. (2017) use the *Earth-Mover* distance, also known as the *Wasserstein-1* distance:

$$W(P, Q) = \inf_{\gamma \in \Pi(P, Q)} \mathbb{E}_{(x, y) \sim \gamma} [| |x - y||_1], \quad (11.85)$$

where $\Pi(P, Q)$ denotes the set of all joint distributions $\gamma(P, Q)$ whose marginals are P and Q , and $| | \cdot ||_1$ stands for the L_1 norm. The reason (11.85) is called the Earth-Mover distance is that the value $\gamma(x, y)$ can be interpreted as the amount of mass that should be transported from x to y to transform distribution P into Q .

In the following example, the EM distance provides a differentiable objective, while JS divergence turns out to be non-differentiable.

Example 11.2 Learning a straight line

A simple example of a situation where the EM metric is differentiable and learnable, but the JS divergence is not was proposed by Arjovsky et al. (2017). To see this, let $z \sim U[0, 1]$ be a uniformly distributed noise source, and θ be a single real-valued parameter. Samples from the model distribution P_θ are obtained as pairs $\theta(\theta, z) \in \mathbb{R}^2$ on a plane. On the other hand, samples from an “expert distribution” P_E are given by pairs $(0, z) \in \mathbb{R}^2$. In other words, distributions P_E and P_θ describe two vertical lines on the plane, where P_E passes through the origin, while P_θ remains the distance $|\theta|$ from the origin. Computing the EM distance (11.85) for this case gives $W(P, Q) = |\theta|$, which is continuous and differentiable for $\theta \neq 0$, and provides a non-vanishing gradient for learning parameter θ . On the other hand, KL divergences $D_{KL}(P_\theta || P_E)$, $D_{KL}(P_E || P_\theta)$, or the JS divergence $D_{JS}(P_\theta, P_E)$ produce vanishing gradients that make it impossible to learn parameter θ (see Exercise 11.6).

Practical tests using Wasserstein GAN in Arjovsky et al. (2017), with generative adversarial training of image recognition networks, have shown significant efficiency improvements of Wasserstein GAN over the conventional GAN with a number of experiments with real-world image datasets.

5.9 Least Squares GAN*

Least squares GAN (LS-GAN) is another interesting extension of GAN that aims at improving its performance in practice. To illustrate the idea of LS-GAN, recall the min–max objective of GAN:

$$\min_G \max_D V_{GAN}(D, G) = \mathbb{E}_{x \sim p_E(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log (1 - D(G(z)))] . \quad (11.86)$$

In the classical GAN, the discriminator $D(x)$ is a classifier that tells samples generated by the expert from those generated by the generator $G(z)$. In this case, the objective of the discriminator in Eq. (11.86) is a cross-entropy objective for binary classification.

The use of the cross-entropy loss for the discriminator $D(x)$ was found in practice to produce problems with training of GAN models. It turns out that this loss function can lead to vanishing gradients for training the generator, leading to numerical instabilities in training. This happens because the cross-entropy loss does not provide sufficient gradients w.r.t. weights for fake examples produced by the generator, when they are on the right side of the decision boundary but are still far from it (Mao et al. 2016).

The least squares GAN (LS-GAN) model (Mao et al. 2016) replaces the cross-entropy discriminator with a least squares loss function. The objective is formulated as a two-step procedure:

$$\begin{aligned} \min_D V_{LSGAN}(D) &= \frac{1}{2} \mathbb{E}_{x \sim p_E(x)} [(D(x) - b)^2] + \frac{1}{2} \mathbb{E}_{z \sim p_z(z)} [(D(G(z)) - a)^2] \\ \min_G V_{LSGAN}(G) &= \frac{1}{2} \mathbb{E}_{z \sim p_z(z)} [(D(G(z)) - c)^2] . \end{aligned} \quad (11.87)$$

Here parameters a and b specify the values (labels) for the fake and real data, respectively, and parameter c gives the value that G wants D to believe in fake data. It turns out that if parameters a, b, c satisfy the constraints $b - c = 1$ and $b - a = 2$, then LS-GAN optimization in Eq. (11.87) is equivalent to minimization of the Pearson χ^2 divergence between the model density and a mixture of the expert and agent densities (see Exercise 11.7). Experiments with image generations have demonstrated improved performance of LS-GAN relatively to the standard GAN for a number of use cases (Mao et al. 2016).

6 Beyond GAIL: AIRL, f-MAX, FAIRL, RS-GAIL, etc.*

Recall that the objective of GAIL is to recover the optimal policy that imitates the expert policy, where the latter is provided via samples from this policy (expert trajectories). The reward function of the agent is *not* learned in GAIL, and this is

why GAIL is a pure imitation learning (IL) algorithm. Indeed, GAIL minimizes the JS distance $D_{JS}(\rho_\pi, \rho_E)$ between the agent's and the expert's occupancy measures using a representation of this measure as a maximum over classifiers $D(s, a)$, see Eq. (11.75). If maximization is performed in a full functional space of *all* admissible classifiers, at the optimum, we would obtain the optimal value

$$D(s, a) = \frac{\rho_E(s, a)}{\rho_\pi(s, a) + \rho_E(s, a)}. \quad (11.88)$$

In practical implementations of GAIL, the discriminator is chosen in some parameterized form $D_w(s, a)$ such as a neural network, so that the optimal value of $D_w(s, a)$ is obtained for a particular vector of model parameters w . Inverting Eq. (11.88) where we substitute $D(s, a) \rightarrow D_w(s, a)$, we have an explicit relation

$$\rho_\pi(s, a) = \frac{1 - D_w(s, a)}{D_w(s, a)} \rho_E(s, a), \quad (11.89)$$

which shows that the optimal discriminator $D_w(s, a)$ should be close to zero at pairs (s, a) generated by the expert policy π_E in order for the measures ρ_π and ρ_E to match at these points. It is important to note here that for numerical implementation all that is expected for a function approximation $D_w(s, a) \in [0, 1]$ is that it should be flexible enough in order to closely approximate its theoretical value (11.88). Other than that, GAIL imposes no further restrictions on the functional form of the discriminator D_w . In particular, there is no link with a reward function—GAIL is a purely imitation learning algorithm that focuses on recovering the optimal policy, but not rewards, from an observed behavior.

It turns out that certain simple modifications of the imitation learning approach of GAIL enable extensions to the original setting of IRL, i.e. to recover *both* the reward and optimal policy. In this section, we will consider a few such extensions.

6.1 AIRL: Adversarial Inverse Reinforcement Learning

Finn et al. (2016) proposed a special functional form for the discriminator function:

$$D(s, a) = \frac{e^{\beta f_\theta(s, a)}}{e^{\beta f_\theta(s, a)} + \pi(a|s)}. \quad (11.90)$$

Here $\pi(a|s)$ is the agent policy, while β and $f_\theta(s, a)$ are, respectively, a parameter and a function whose meaning will be clarified momentarily.

We define the reward function $\hat{r}(s, a)$ as the rescaled log-odds of the classifier $D(s, a)$:

$$\hat{r}(s, a) = \frac{1}{\beta} \log \frac{D(s, a)}{1 - D(s, a)} = f_\theta(s, a) - \frac{1}{\beta} \log \pi(a|s). \quad (11.91)$$

Using these rewards for random trajectories, the value function $\hat{V}(s)$ is given by a discounted expected value of all future rewards:

$$\begin{aligned}\hat{V}(s) &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \hat{r}(s_t, a_t) \middle| s_0 = s \right] \\ &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \left(f_{\theta}(s, a) - \frac{1}{\beta} \log \pi(a|s) \right) \middle| s_0 = s \right].\end{aligned}\quad (11.92)$$

Respectively, policy optimization amounts to maximizing the value function (11.92).

We can compare this with Eq. (11.24) that we used for the free energy (entropy-regularized value function) in the setting of Maximum Causal Entropy IRL. These expressions produce identical policies if we set $f_{\theta}(s, a) = r_{\theta}(s, a) + g(s)$, where $g(s)$ is an arbitrary function of state. As we will see shortly, at the optimum $f_{\theta}(s, a)$ coincides with the advance function, providing an interpretation of the function $g(s)$.

The adversarial inverse reinforcement learning (AIRL) model (Finn et al. 2016; Fu et al. 2015) amounts to using the special form (11.90) of the discriminator in the GAIL objective (11.81), where optimization of a general function $D(s, a)$ is replaced by optimization of parameters θ of function $f_{\theta}(s, a)$:

$$\begin{aligned}J(\pi, \theta) &= \min_{\pi} \max_{\theta} \mathbb{E}_{\pi_E} \left[\log \frac{e^{\beta f_{\theta}(s, a)}}{e^{\beta f_{\theta}(s, a)} + \pi(a|s)} \right] \\ &\quad + \mathbb{E}_{\pi} \left[\log \frac{\pi(a|s)}{e^{\beta f_{\theta}(s, a)} + \pi(a|s)} \right] - \lambda H^{causal}(\pi).\end{aligned}\quad (11.93)$$

The reader can verify (see Exercise 11.8) that gradients of this objective with respect to parameters θ match gradients of an importance sampling method for the MaxEnt IRL (see Eq. (11.38)).

AIRL is performed by iterating between steps of optimizing the discriminator with respect to parameters, and optimizing the policy by maximizing the RL objective $\max_{\pi} \mathbb{E}_{\tau \sim \pi} [\gamma^t \hat{r}(s_t, a_t)]$, see Eq. (11.92). The global minimum of the discriminator objective is achieved when $D = \frac{1}{2}$. At this point, using Eq. (11.90) we obtain

$$\pi(a|s) = e^{\beta f_{\theta}(s, a)}. \quad (11.94)$$

As the policy $\pi(a|s)$ should be normalized to one, this means that we should have $\int e^{\beta f_{\theta}(s, a)} da = 1$.

Let us compare this with the policy (11.30) of Maximum Causal Entropy IRL for a special case of a uniform reference policy $\pi_0(a|s)$, which we can write as follows:

$$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t) = \frac{1}{Z_\theta(\mathbf{s}_t)} e^{\beta G_\theta(\mathbf{s}_t, \mathbf{a}_t)}, \quad Z_\theta(\mathbf{s}_t) := \int e^{\beta G_\theta(\mathbf{s}_t, \mathbf{a}_t)} d\mathbf{a}_t. \quad (11.95)$$

This policy can be equivalently written as follows:

$$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t) = e^{\beta * (G_\theta(\mathbf{s}_t, \mathbf{a}_t) - \frac{1}{\beta} \log Z_\theta(\mathbf{s}_t))} := e^{\beta A_\theta(\mathbf{s}_t, \mathbf{a}_t)}, \quad (11.96)$$

where

$$A_\theta(\mathbf{s}_t, \mathbf{a}_t) := G_\theta(\mathbf{s}_t, \mathbf{a}_t) - \frac{1}{\beta} \log \int e^{\beta G_\theta(\mathbf{s}_t, \mathbf{a}_t)} d\mathbf{a}_t = G_\theta(\mathbf{s}_t, \mathbf{a}_t) - V_\theta(\mathbf{s}_t) \quad (11.97)$$

is the *advantage function*. Comparing Eq. (11.96) with Eq. (11.94), we find that $f_\theta^\star(s, a) = A_\theta^\star(s, a)$, where A_θ^\star is the optimal advantage function. This means that at the optimum discriminator, AIRL learns the advantage function (Fu et al. 2015). The latter can be used to recover one-step rewards.

The AIRL method can also be extended to learn not only the reward function, but also the shaping function (see Eq. (11.4)). When the shaping function is learned alongside the reward, results of such learning can be transferred to a new environment with dynamics different from those used for training (Fu et al. 2015).

6.2 Forward KL or Backward KL?

As we noted in Sect. 5.7, both the KL divergence and JS divergence are special cases of a wider class of f-divergences. In particular, if we take $f(x) = -\log x$, we obtain

$$D_f(\rho_E(s, a) || \rho_\pi(s, a)) = D_{KL}(\rho_\pi(s, a) || \rho_E(s, a)). \quad (11.98)$$

This divergence is often referred to as the “reverse” KL divergence.

On the other hand, if we take $f(x) = x \log x$, we obtain

$$D_f(\rho_E(s, a) || \rho_\pi(s, a)) = D_{KL}(\rho_E(s, a) || \rho_\pi(s, a)). \quad (11.99)$$

This divergence is called the “forward” KL divergence.

Given that both the “backward” and “forward” KL divergences are obtained as special cases of f-divergence, and that they both evaluate how measure ρ_π is different from the “expert” measure ρ_E , one may wonder which one of these measures, if any, should be preferred for tasks of imitation learning.

As it turns out, all types of imitation learning (IL) amount to minimization of some statistical divergence of measures ρ_π and ρ_E . For example, in BC, we minimize the expected *forward* KL divergence:

$$\mathcal{L}_{BC} = \mathbb{E}_{\rho_E} [KL(\rho_E(s, a) || \rho_\pi(s, a))] = -\mathbb{E}_{\rho_E} [\log \rho_\pi(s, a)] - \mathcal{H}_{\rho_E}(s, a). \quad (11.100)$$

Since the entropy $\mathcal{H}_{\rho_E}(s, a)$ of the expert policy is independent of the agent policy $\pi_\theta(a|s)$, this term can be dropped. Minimization of the first term is exactly equivalent to minimization of the negative log-likelihood in maximum likelihood method.

The forward and backward KL divergences exhibit a different behavior for a learned policy π . While the forward KL divergence encourages a behavior that matches π and π_E only on average (as this is essentially what maximum likelihood method does), minimization of the backward KL divergence enforces a “mode-seeking” behavior of π , which tries to match the expert policy better in terms of most plausible actions.

Example 11.3 Forward vs backward KL: mode-covering vs mode-seeking

Here we illustrate the key differences between optimization of the forward KL divergence

$$D_{KL}(\rho_E(s, a) || \rho_\pi(s, a)) = \int \rho_E(s, a) \log \frac{\rho_E(s, a)}{\rho_\pi(s, a)} ds da \quad (11.101)$$

and the backward KL divergence

$$D_{KL}(\rho_\pi(s, a) || \rho_E(s, a)) = \int \rho_\pi(s, a) \log \frac{\rho_\pi(s, a)}{\rho_E(s, a)} ds da. \quad (11.102)$$

In both cases, the problem is to find policy π that minimizes (respectively, the forward or backward) KL divergence with the expert policy π_E .

Let us first consider the forward KL divergence (11.101). In this case, the expert density serves as set of weights. Minimization of the forward KL therefore tries to match π with π_E wherever $\pi_E > 0$. In other words, π tries to match π_E everywhere, as much as it can. This is called the *mode-covering* behavior.

Now consider the backward KL divergence (11.102). In this case, it is the agent’s policy π that serves as weights in minimization. If we set π to zero at some values of a , where π_E is non-zero, there is no penalty for this using the backward KL divergence. This means that with this method, an agent’s policy would try to match the expert policy *only* at some region of all trajectories, rather than on *all* trajectories, as would be the case with the forward KL divergence. In other words, minimization of the backward KL divergence enforces the *mode-seeking* behavior.

(continued)

Example 11.3 (continued)

To illustrate these differences, let us assume that the expert policy π_E is bimodal, but we try to approximate it with a parameterized policy π_θ that only allows for uni-modal distributions. What sort of policies would be picked by both KL divergences?

Minimization of the forward KL divergence tries to cover all observations as much as it can, so it would put a uni-modal density some way in between of two maxima of the bimodal expert policy.

On the other hand, minimization of the backward KL divergence will produce a uni-modal policy that will try to match the largest component of the expert's bimodal policy. Assume that the largest component in the expert policy corresponds to optimal actions while the smaller component corresponds to sub-optimal actions. In this case, we would say that the backward KL does the right job—it focuses at optimal actions, instead of spreading over both optimal and sub-optimal actions demonstrated by the expert.

While BC minimizes the *forward* KL divergence, as we will see next, both AIRL and some of its extensions minimize the *backward* KL divergence.

6.3 f-MAX

Generalizations of AIRL can be obtained using other f-divergences. The f-MAX method (Chasemir et al. 2019) is based on optimization of the f-divergence $D_f(\rho_E(s, a) || \rho_\pi(s, a))$. This is done using the following iterative optimization procedure (see Eq. (11.84) in Sect. 5.7):

$$\begin{aligned} \max_{T_\omega} F(\theta, \omega) &= \mathbb{E}_{x \sim P_{\pi_E}} [T_\omega] - \mathbb{E}_{x \sim \pi} [f^*(T_\omega)] \\ \max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_t f^*(T_\omega) \right]. \end{aligned} \quad (11.103)$$

Here the first equation minimizes the f-divergence $D_f(\rho_E(s, a) || \rho_\pi(s, a))$ by optimizing T_ω . The policy optimization objective is equivalent to minimizing the first equation in π , or equivalently maximizing the second equation. These equations produce the f-MAX method for a general f-divergence.

Now assume we choose the *backward* KL divergence (11.98) as a special case of f-divergence with $D_f(\rho_E(s, a) || \rho_\pi(s, a)) = D_{KL}(\rho_\pi(s, a) || \rho_E(s, a))$. This corresponds to the choice $f(x) = -\log x$. The convex dual for this case is $f^*(y) = -1 - \log(-y)$ and $T_\omega^\pi(s, a) = -\frac{\rho_\pi(s, a)}{\rho_E(s, a)}$ (Nowozin et al. 2016). Substituting these expressions in the second of Eqs. (11.103), the latter is restated as follows:

$$\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_t f^*(T_{\omega}^{\pi}(s_t, a_t)) \right] = \max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_t \log \rho_E(s_t, a_t) - \log \rho_{\pi}(s_t, a_t) - 1 \right]. \quad (11.104)$$

On the other hand, the policy objective of AIRL can also be expressed by substituting the optimal discriminator (11.88) into Eq. (11.92) with \hat{r} defined in Eq. (11.91). This gives

$$\begin{aligned} \hat{V}(s) &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \hat{r}(s_t, a_t) \middle| s_0 = s \right] = \frac{1}{\beta} \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t \log \frac{D(s_t, a_t)}{1 - D(s_t, a_t)} \middle| s_0 = s \right] \\ &= \frac{1}{\beta} \mathbb{E}_{\tau \sim \pi} \left[\sum_t \log \rho_E(s_t, a_t) - \log \rho_{\pi}(s_t, a_t) \right]. \end{aligned} \quad (11.105)$$

This equation differs from Eq. (11.104) only by unessential additive and multiplicative constants; therefore, their maximization gives rise to identical solutions. This shows that AIRL solves the MaxEnt IRL problem by minimizing the *reverse* KL divergence (Chasemir et al. 2019).

This example shows that AIRL can be considered a special case of a more general f-MAX algorithm (11.103) that generalizes it to more general class of f-divergences.

Furthermore, as shown in Chasemir et al. (2019), f-MAX can also be considered a subset of the cost-regularized MaxEnt IRL framework of Ho and Ermon, given in the general form by Eq. (11.70). To this end, we take the following cost regularizer:

$$\psi_f(c) = \mathbb{E}_{\rho_E} [f^*(c(s, a)) - c(s, a)]. \quad (11.106)$$

With this choice, we obtain for the convex conjugate

$$\psi_f^*(\rho_{\pi}(s, a) - \rho_E(s, a)) = D_f(\rho_{\pi}(s, a) || \rho_E(s, a)), \quad (11.107)$$

which produces a generalized Ho-Ermon objective

$$\begin{aligned} \text{RL} \circ \text{IRL}(\pi_E) &= \min_{\pi} -H^{causal}(\pi) + \psi^*(\rho_{\pi} - \rho_E) \\ &= \min_{\pi} -H^{causal}(\pi) + D_f(\rho_{\pi}(s, a) || \rho_E(s, a)). \end{aligned} \quad (11.108)$$

6.4 Forward KL: FAIRL

While f-MAX is a general method that supports many occupancy metric distances, it turns out that it does not work for the forward KL divergence. There are a few reasons for this property. First, forward KL corresponds to the following specifications:

$$f(x) = x \log x, \quad f^*(y) = \exp(y - 1), \quad T_\omega^\pi = 1 + \log \frac{\rho_E(s, a)}{\rho_\pi(s, a)}. \quad (11.109)$$

With this choice, the objective of the policy optimization (the second of Eqs. (11.103)) becomes

$$\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_t f^*(T_\omega) \right] = \mathbb{E}_{\rho_\pi} \left[\frac{\rho_E(s, a)}{\rho_\pi(s, a)} \right] = 1. \quad (11.110)$$

This means that no signal is provided for training the policy with the choice of forward KL divergence.

To produce a usable algorithm for optimizing forward KL divergence, let us consider a simple modification of the AIRL reward function which produces a forward KL counterpart of AIRL. Recall that AIRL chooses the following form of the reward function:

$$\hat{r}(s, a) = \frac{1}{\beta} \log \frac{D(s, a)}{1 - D(s, a)}. \quad (11.111)$$

Instead of this choice, FAIRL (“forward-AIRL”) (Chasemaniour et al. 2019) uses a different definition of the one-step reward:

$$\hat{r}(s, a) = \frac{1}{\beta} \frac{D(s, a)}{1 - D(s, a)} \log \frac{1 - D(s, a)}{D(s, a)}. \quad (11.112)$$

As can be checked, for this choice we have

$$\mathbb{E}_{\tau \sim \pi} \left[\sum_t \gamma^t \hat{r}(s_t, a_t) \right] \sim -D_{KL}(\rho_E(s, a) || \rho_\pi(s, a)). \quad (11.113)$$

Therefore, in FAIRL, policy optimization with the reward defined in Eq. (11.112) is equivalent to minimization of the forward KL divergence. As we mentioned above, forward KL divergence promotes a mode-covering behavior, as opposed to a mode-seeking behavior that is obtained with AIRL. In certain situations, this can turn into an advantage, as was found in experiments with simulated robots (Chasemaniour et al. 2019).

? How to Relate the Reward and the Discriminator?

AIRL and FAIRL provide two examples of a reward as a function of the discriminator, expressed, respectively, in Eqs. (11.111) and (11.112). Both

(continued)

choices ensure the property that the reward increases when D increases, which enforces the assumption of near-optimality of expert's actions.

We may ask if there are other useful ways to relate the reward and the discriminator in the setting of adversarial imitation learning. To develop some intuition, consider the following example. We assume that rewards are bounded and are described by a logistic equation

$$r(s, a) = \frac{1}{1 + \exp(-\theta^T \Psi(s, a))}, \quad (11.114)$$

where $\theta = (\theta_1, \dots, \theta_K)$ is a vector of parameters, and $\Psi(s, a) = (\Psi_1(s, a), \dots, \Psi_K(s, a))$ is a set of K pre-specified basis functions. The discriminator is a function of the same features (s, a) , and is assumed to have a similar form:

$$D(s, a) = \frac{1}{1 + b \exp(-\theta^T \Psi(s, a))}, \quad (11.115)$$

where b is a parameter. Simple algebraic manipulation allows us to express $r(s, a)$ in terms of $D(s, a)$:

$$r = r(D(s, a)) = \frac{D(s, a)}{b + (1 - b)D(s, a)}. \quad (11.116)$$

Evidently, this choice is similar to parameterizations of AIRL and FAIRL in terms of monotonicity of a relation between D and r , except that in Eq. (11.116), the reward is bounded, and satisfies $r \rightarrow 0$ when $D \rightarrow 0$, and $r \rightarrow 1$ when $D \rightarrow 1$. If D is considered to be the tail probability (a.k.a. the *decumulative distribution function*), then a bounded reward $0 \leq r(D) \leq 1$ can be considered as a transformation of the ddf for the discriminator that produces another ddf. This can be used to construct more general models than Eq. (11.116). For example, parameter b in Eq. (11.116) can be made a function $b(s, a)$.

6.5 Risk-Sensitive GAIL (RS-GAIL)

As we discussed in previous chapters on reinforcement learning, in the standard formulation of RL, one maximizes the expected total reward from a trajectory. This trajectory is given by a sum of all one-step rewards. This is called the *risk-neutral*

approach: the standard RL depends on the mean of the total reward, but not on its higher order moments, e.g. its variance or tail probabilities. On the other hand, for financial applications, risk is often an integral and critical part of a decision-making process such as trading certain positions in a portfolio. Outside of finance, risk-sensitive imitation learning is often desirable for human and robot safety, and thus is a topic of active research in the machine learning community.

In certain cases, simple modifications of the reward in the traditional risk-neutral RL approach enable incorporating some measures of *risk* from taking actions. For example, in the QLBS model of reinforcement learning for option pricing that we presented in Chap. 10, one-step rewards incorporate risk penalties proportional to the variance of the option hedging portfolio.

A more general approach to risk-sensitive imitation learning is to assume that the agent tries to minimize the loss C_π of policy π with respect to the expert cost function c :

$$\min_{\pi} \mathbb{E}[C_\pi], \quad (11.117)$$

subject to the constraint

$$\rho_\alpha [C_\pi] \leq \rho_\alpha [C_{\pi_E}]. \quad (11.118)$$

Here ρ_α is some risk measure, for example, CVaR at the confidence level α . The meaning of the constraint (11.118) is that the agent should at most have the same risk as the expert. Adding the constraint to the objective using the Lagrange multiplier method produces the equivalent unconstrained problem

$$\min_{\pi} \max_{\lambda \geq 0} \mathbb{E}[C_\pi] - \mathbb{E}[C_{\pi_E}] + \lambda (\rho_\alpha [C_\pi] - \rho_\alpha [C_{\pi_E}]). \quad (11.119)$$

As the cost is unobserved by the agent, we maximize this expression over all cost functions $c(s, a) \in C$, with $C_\pi(c)$ being the loss of policy π with respect to the cost function $c(s, a)$. This produces the following min–max problem:

$$\min_{\pi} \max_{c \in C} \max_{\lambda \geq 0} \mathbb{E}[C_\pi(c)] - \mathbb{E}[C_{\pi_E}(c)] + \lambda (\rho_\alpha [C_\pi(c)] - \rho_\alpha [C_{\pi_E}(c)]). \quad (11.120)$$

Finally, we swap minimization over π with maximization over λ , and add, similarly to GAIL, the causal entropy and a convex regularizer $\psi(c)$ to the objective. This gives rise to the risk-sensitive GAIL (RS-GAIL) algorithm (Lacotte et al. 2018):

$$\begin{aligned} & \max_{\lambda \geq 0} \min_{\pi} -H^{causal}(\pi) + \mathcal{L}_\lambda(\pi, \pi_E) \\ & \mathcal{L}_\lambda(\pi, \pi_E) := \max_{c \in C} (1 + \lambda) (\rho_\alpha^\lambda [C_\pi(c)] - \rho_\alpha^\lambda [C_{\pi_E}(c)]) - \psi(c), \end{aligned} \quad (11.121)$$

where $\rho_\alpha^\lambda [C_\pi(c)] := (\mathbb{E}[C_\pi] + \lambda\rho_\alpha[C_\pi])/(1+\lambda)$ is the coherent risk measure for policy π for mean-CVaR with the risk parameter λ . The parameter λ controls the tradeoff between the mean performance and risk of the policy. The authors of Lacotte et al. (2018) produced two versions of their algorithm by using regularizers $\psi(c)$ that yield either the JS or Wasserstein metrics between ρ_π and ρ_E . They found improved performance compared to GAIL, in terms of both mean rewards and risk metrics, for a number of simulated robot environments.

6.6 Summary

In this section, we outlined a variety of extensions of generative adversarial imitation learning (GAIL), which use different metrics instead of the JS divergence used by GAIL, in order to learn both the optimal policy and reward function. As all imitation learning algorithms amount to minimization of some metric between occupancy measures induced by the action and the expert, exploring alternative formulations by choosing different regularizers is helpful for exploring efficiency of imitation learning for different environments. Models presented in this section cover both the traditional risk-neutral setting of RL/IRL and risk-sensitive approaches. To reiterate, most of the RL/IRL problems that we encounter in finance require some control of risk inherent in decision-making, which makes risk-sensitive approaches especially interesting for financial applications.

Inverse reinforcement learning and imitation learning are actively pursued by researchers in the machine learning community. While the main activity in these direction seems to be largely driven by applications to robotics and video games, there is also research into applications of methods of IRL and IL in neuroscience, marketing, consumer research, and, last but not least, in finance. Before turning to financial applications of IRL, in the next few sections, we consider other methods for IRL that show promising utility in financial applications.

7 Gaussian Process Inverse Reinforcement Learning

The IRL methods presented thus far in this chapter describe probabilistic models which treat the (unknown) reward function as a fixed deterministic function of features (or, equivalently, basis functions). Given a representative set of K basis functions of states and actions $\Psi_k(s, a)$ with $k = 1, \dots, K$, one common approach is to construct a reward function as a linear expansion in features $r(s, a) = \sum_k \theta_k \Psi_k(s, a)$. We assume that basis functions are bounded, so that the resulting

reward would also be bounded for finite coefficients θ_k .⁸ The problem of learning the reward reduces to the problem of finding a finite set of coefficients θ_k .

While conceptually simple, this approach has obvious limitations, as it requires a pre-defined set of “good” basis functions. Constructing a simple and manageable set of basis functions is relatively straightforward in low-dimensional continuous state-action spaces. For example, for one-dimensional and bounded state and action spaces, we could simply construct two individual sets of basis functions (e.g., B-splines) for both state and action spaces, and then take their direct product as a set of basis functions in the state-action space. If both sets have N basis functions, we end up with N^2 basis functions for the joint state-action space. For higher dimensional cases, such an approach based on using direct products of individual bases would clearly be problematic, as it would lead to exponential growth of both the dimension of the basis and the number of parameters to be estimated from data.

When the choice of basis functions (features) for IRL is not obvious, one possible alternative approach is to use demonstrations to learn *both* the features and the reward function. Note that the problem of learning high-level features from data is a classical problem in supervised and unsupervised learning. For example, when deep learning architectures are used for classification tasks, inner layers of a network construct higher-level features out of raw data. Importantly, in supervised learning there is a clear criterion for assessing the quality of such learned features—it is measured by the quality of the classifier itself, and is learned through back-propagation. In the setting of RL and IRL, this is the path followed, respectively, by Deep RL and Deep IRL approaches.

In this section, we will consider different Bayesian approaches to IRL. Unlike MaxEnt IRL that uses deterministic rewards and stochastic policies, in Bayesian reinforcement learning, the reward is random, and has a certain probability distribution on the state-action space. As in IRL, the reward is unobserved and can be modeled with a Bayesian approach using a hidden (latent) variable. The objective of Bayesian IRL is then to learn the probability distribution of this hidden variable from observed data.

7.1 Bayesian IRL

Let us assume that $r = r(s, a)$ is a random variable whose value may depend on states and actions. We will collectively denote them in this section as $X = (s, a)$. Let $p(r|X)$ be a prior probability distribution for the random reward given features X . This distribution presents our views of rewards in different states of the world that we hold prior to observing data \mathcal{D} consisting of sequences of states and actions from expert demonstrations.

⁸Alternatively, we can consider unbounded basis functions but bounded non-linear functional specifications of the reward function.

For a fixed reward function r , the probability of observing data \mathcal{D} is given by some likelihood function $p(\mathcal{D}|r, \theta)$, where θ are adjustable model parameters. While the likelihood function can be explicitly computed using specific RL models, here we temporarily leave it unspecified.

The joint probability to observe data \mathcal{D} and rewards r given input features X is

$$p(\mathcal{D}, r|X) = p(r|X) p(\mathcal{D}|r, \theta). \quad (11.122)$$

In IRL, we do not observe rewards; therefore, we form the expected likelihood of data by integrating over all possible values of r in Eq. (11.122). This produces

$$p(\mathcal{D}|X) = \int p(r|X) p(\mathcal{D}|r, \theta) dr. \quad (11.123)$$

To maximize the likelihood of observed expert data, this marginalized probability should be maximized with respect to parameters θ . However, this expression involves the integral over all attainable values of rewards. To compute it numerically, this integral could be discretized to a set of M possible values of rewards. But this would imply that we have to solve M direct reinforcement problems of computing $p(\mathcal{D}|r, \theta)$ in order to solve one IRL problem in Eq. (11.123).

As an alternative to a direct calculation of the integral (11.123) using discretization, we could alternatively estimate it using the Laplace (saddle-point) approximation. The latter approximation applies when the integrand in Eq. (11.123) has a strong peak around a certain value r_* of the argument. The saddle-point approximation is then obtained by expanding $\log(p(r|X) p(\mathcal{D}|r, \theta))$ in a low-order Taylor expansion around r_* , and performing Gaussian integration over deviations $\Delta r = r - r_*$.

Both these approaches assume a given parametric prior distribution $p(r|X)$ of rewards. In certain applications, e.g. in robotics, it is sometimes tedious or cumbersome to devise such a parametric prior. Instead of using parametric priors, Gaussian process (GP) IRL operates with very flexible *non-parametric* specifications of the reward functions by operating with distributions over such functions.

7.2 Gaussian Process IRL

In the Gaussian process (GP) approach, the prior reward distribution is modeled as a zero-mean GP prior:

$$r \sim \mathcal{GP}(0, k_\theta(\mathbf{x}_i, \mathbf{x}_j)). \quad (11.124)$$

Here k_θ stands for the covariance function, e.g.

$$k_\theta(\mathbf{x}_i, \mathbf{x}_j) = \sigma_k^2 e^{-\frac{\xi}{2} (\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j)} \quad (11.125)$$

and $\theta = (\sigma_k, \xi)$ is the vector of model parameters.

For a finite sample of data, the GP prior induces the probability $r \sim \mathcal{N}(0, K_{XX})$, where K_{XX} is the covariance matrix with elements $[K_{XX}]_{ij} = k_\theta(\mathbf{x}_i, \mathbf{x}_j)$. Learning of the reward amounts to learning parameters θ of the kernel function (11.125).

Recall from Chap. 3 that in GP regression, the objective is to evaluate values of an unknown function without committing to a particular parametric form of this function, given inputs x and values f of function $f(x)$ at these points. The task is to find the posterior distribution f_{star} on test data points x_* . As the joint distribution of f and f_* is Gaussian, the posterior distribution of f_* is also a Gaussian:

$$f_*|x, x_*, f \sim \mathcal{N}\left(K_{x_*,x} K_{x,x}^{-1} f, K_{x_*,x_*} - K_{x_*,x} K_{x,x}^{-1} K_{x,x_*}\right). \quad (11.126)$$

In GPIRL, unlike GP regression, values of the function sought (i.e., the reward function) are not observed. To link with observations, GPIRL uses MaxEnt IRL (Levine et al. 2011). Let u denote the true reward, while r denotes its noisy version. The posterior probability of u and θ given data \mathcal{D} and a set of “inducing points” X_u is

$$P(u, \theta | \mathcal{D}, X_u) P(\mathcal{D}, u, \theta | X_u) = P(u, \theta | X_u) \left[\int P(\mathcal{D}|r) P(r|u, \theta, X_u) dr \right]. \quad (11.127)$$

In this expression, $P(\mathcal{D}|r)$ is the probability of observations conditioned on a fixed reward. To evaluate this expression, GPIRL uses a MaxEnt policy which involves the exponential dependence on r . To compute the integral over r , it can be discretized, or alternatively estimated using the saddle-point approximation. The limiting case of the saddle-point approximation was used by Levine et al. (2011) who considered the limit of zero noise in r , which makes $P(r|u, \theta, X_u)$ the delta-function $\delta(r - u)$.

8 Can IRL Surpass the Teacher?

All algorithms for IRL and imitation learning presented so far in this chapter share the same common assumption that trajectories (behavior) demonstrated by a teacher are optimal or nearly optimal. This assumption makes the problem somewhat easier in the sense that if the agent knows that demonstrated behavior is nearly optimal, all it needs to do is to *imitate* the teacher, rather than infer the teacher’s intention.

However, this assumption might be too stringent or unrealistic in many cases of practical interest. For example, in teaching robots from human demonstrations, it is not always easy to measure or control the level of sub-optimality of a demonstrator. A combination of imitation learning (instead of learning of intent) with a possible sub-optimality of demonstrations can lead to less controllable or understandable learning policies and rewards.

Another potential objection to the standard paradigm of learning only from optimal or nearly optimal demonstrations is that having access to failed (or strongly non-optimal) demonstrations can often be very informative of the teacher’s goals. For example, in our financial cliff walking example, the conventional IRL can be trained using only high reward trajectories that do not fall from the cliff to the bankruptcy state. Intuitively, we can expect that learning can be improved by showing the agent failed trajectories that end up in the bankruptcy state, and passing them to a policy optimization module as trajectories to be avoided. This would be more similar to human learning, yet most of the existing IRL or IL approaches do not incorporate such information, which may lead to less efficient and more data-intensive algorithms. In this section, we present a powerful extension to MaxEnt IRL that incorporates learning from both successful and failed demonstrations.

To understand the implications of this approach in financial applications, let us consider introducing an agent that learns from a human investor or human trader. As the human is subject to behavioral biases and takes sub-optimal or wrong trading decisions from time to time, the best the agent that follows the standard IRL or IL approach can do is to build an “AI alter ego” of the human that propagates such errors and biases. But ideally we would not just want to *imitate* the trader’s strategy, but rather to *improve* it by inferring the actual trader’s intentions and optimizing a policy that captures these “true” intentions. Such an approach, of course, is the tip of the iceberg in human–machine interaction. See Capponi et al. (2019) for an example of human–machine interaction in the context of robo-advisory.

To surpass performance in demonstrations, an agent should learn from a variety of demonstrations including not only optimal or nearly optimal ones, but also severely sub-optimal or outright failed demonstrations. Indeed, the task of surpassing rather than imitating a teacher amounts to *extrapolation* in a decision space, but such extrapolation might be impossible, or would generalize poorly, if all demonstrations are (nearly) optimal.

In this section we consider a few recent ideas in the machine learning literature which aim at surpassing a teacher by capturing teacher’s intents rather than simply imitating the behavior.

8.1 *IRL from Failure*

In many situations, providing high-quality expert data for training an IRL algorithm can be costly or problematic. Consider, for example, a possible application of IRL to mimic a human trader. Particular observed sequences of trader’s actions may not be necessarily optimal, even from the point of view of internal goals of the trader. Therefore, in this case, it would be difficult to quantify the amount of optimality in demonstrated trajectories. On the other hand, traditional IRL schemes such as MaxEnt IRL generally assume that demonstrated trajectories are close to optimal ones, with only occasional deviations from the optimal behavior. This means that

a successful IRL model can at best only *justify* the behavior observed in expert demonstration, rather than *explain* it.

The above example implies that insistence on optimality of demonstrations may not always be feasible. But interestingly, this may not even be desirable for many problems. Just as in human teaching, which provides examples of both desired and undesired behavior, providing an artificial agent with information about undesired behavior (policies) can be helpful with identifying optimal policies from demonstrations using data that contains both successful and unsuccessful demonstrations. Unsuccessful demonstrations might also be easier to produce than successful ones.

The IRL from Failure (IRLF) model (Shiarlis et al. 2016) suggests an extension of the “classical” Max-Causal Entropy IRL that operates with both successful and failed demonstrations. The latter two sets of demonstrations are referred to as \mathcal{D} and \mathcal{F} , respectively. The central idea of the IRLF model is to combine two criteria. First, feature expectations of the learned policy should match their empirical expectations. This part is the same as in the conventional MaxEnt IRL. The additional objective that is new to IRLF is to also require these feature expectations be maximally *dissimilar* to the empirical expectations of \mathcal{F} . This may facilitate learning of the reward that simultaneously encourages state-action pairs observed in \mathcal{D} and discourages those found in \mathcal{F} .

More formally, assume there are K features $\phi_k(s, a)$, and rewards are linearly parameterized in terms of these features as $r(s, a) = (w^{\mathcal{D}} + w^{\mathcal{F}})^T \phi(s, a)$, where $w^{\mathcal{D}}$ and $w^{\mathcal{F}}$ are two parameter vectors of length K whose meaning will be clarified momentarily. Let $\tilde{\mu}_k^{\mathcal{D}}$ and $\tilde{\mu}_k^{\mathcal{F}}$ be empirical feature expectations for \mathcal{D} and \mathcal{F} , respectively, and μ_k^{π} be feature expectations under policy π . The IRLF model maximizes the following objective function:

$$\begin{aligned} J(\pi, \theta, z, w^{\mathcal{D}}, w^{\mathcal{F}}) &= H(A||S) - \frac{\lambda}{2} \|\theta\|^2 + \sum_{k=1}^K \theta_k z_k \\ &\quad + \sum_{k=1}^K w_k^{\mathcal{D}} (\mu_k^{\pi}|_{\mathcal{D}} - \tilde{\mu}_k^{\mathcal{D}}) + \sum_{k=1}^K w_k^{\mathcal{F}} (\mu_k^{\pi}|_{\mathcal{F}} - \tilde{\mu}_k^{\mathcal{F}} - z_k). \end{aligned} \quad (11.128)$$

Here $H(A||S)$ stands for the causal entropy, π is the optimal policy, and $w^{\mathcal{D}}, w^{\mathcal{F}}$ are two sets of feature expansion coefficients that serve as Lagrange multipliers for two classes of constraints. The first constraint is $\mu_k^{\pi}|_{\mathcal{D}} = \tilde{\mu}_k^{\mathcal{D}}$ ensuring matching successful trajectories using policy π . The second constraint reads $\mu_k^{\pi}|_{\mathcal{F}} - \tilde{\mu}_k^{\mathcal{F}} = z_k$, and involves auxiliary variables $z_k \in \mathbb{R}$. The objective (11.128) is maximized over variables z_k alongside with maximization with respect to parameters $\theta, w^{\mathcal{D}}, w^{\mathcal{F}}$ and policy π . Maximization with respect to z_k in Eq. (11.128) achieves the goal of producing feature expectations that would be dissimilar to empirical features on failed trajectories. It is also computationally convenient as the final IRLF formulation (11.128) amounts to convex optimization, see Shiarlis et al. (2016) for details.

The IRLF method works in a similar way to MaxEnt IRL, iterating between updating the reward function $r(s, a) = (w^{\mathcal{D}} + w^{\mathcal{F}})^T \phi(s, a)$ and policy updates via soft Q-iteration. Though the number of parameters in $w^{\mathcal{D}}, w^{\mathcal{F}}$ is twice larger than for the conventional MaxEnt IRL, update of parameters $w^{\mathcal{F}}$ can be performed analytically, so that the computational overhead in comparison to the standard MaxEnt IRL is minimal. An interesting additional property of the IRLF model is that it also handles cases where failed trajectories are similar to successful trajectories, with respect to some of their features, and dissimilar with respect to others.

8.2 Learning Preferences

According to the standard RL approach, a reward function gives the most succinct description of a demonstrator’s objectives. Respectively, the conventional IRL approach focuses on learning the reward function from a set of demonstrations. However, as we discussed earlier in this chapter, inference of the reward from demonstrations is an ill-posed inverse problem that produces an infinite number of solutions, unless a regularization is imposed on the solution (e.g., as it is done in MaxEnt IRL). Furthermore, rewards can only be found up to arbitrary linear rescaling and additive shaping transformations, which complicates transfer of learned rewards and policies across different environments.

An alternative to learning reward functions as the most condensed expression of a teacher’s goals is to instead learn the *preferences* of the teacher. Preferences can be viewed as an alternative representation of the intentions of the teacher. Unlike rewards that are defined for a single state-action combination, preferences are specified for *pairs* of state-action combinations, or alternatively for pairs of multi-step trajectories.

Preferences are formulated as rank-ordering (ordinal) relations that may or may not rely on any quantitative measure of goodness of separate actions or trajectories. Clearly, if one works with a reward function whose values are known for all state-action pairs, any pair of trajectories can be rank-ordered based on accumulated rewards. However, preference relations can also be established between trajectories without specifying numerical rewards using only an ordinal measure. This is somewhat similar to corporate bond ratings, e.g. AAA, AA, A, etc. The default probability for a bond rated AA is assumed to be lower than for a A-rated bond; however, ratings themselves do not convey information about the absolute levels of default risk in different ratings categories.

Conceivably, there are many situations where defining pairwise-based qualitative ranking over demonstrations might be easier or more intuitive than directly assigning a numerical reward value to different individual trajectories. For example, a portfolio manager could be more at ease by expressing relative rankings of different stock purchase decisions rather than expressing her satisfaction from these trades in terms of a single reward value.

It is therefore of interest to consider the problem of *preference-based IRL* where, given a set of ranked trajectories, the objective is to infer why some trajectories are better than others, i.e. to understand the *intent* of a demonstrator. The end result of such inference is again compressed into a reward function, as in the conventional IRL approaches. However, as this reward is based on inferred intent rather than simply on mimicking demonstrator’s behavior, it can be further used with the conventional RL approaches to learn policies that can *improve* the performance in demonstrations, rather than simply mimic it, as in traditional IRL approaches.

Preference-based IRL can also be considered a natural extension of the approach taken in the IRLF model. In the latter, all demonstrations are partitioned into either successful and failed, without further specifying the amount of success or failure. With preference-based IRL, one operates with a range of demonstrations of varying quality, which are then rank-ordered using some success criterion (e.g., using a hidden reward of demonstrators). This can provide the IRL agent with more refined information than a simple binary classification into the successful and failed trajectories.

8.3 T-REX: Trajectory-Ranked Reward EXtrapolation

As we mentioned above, to improve upon the performance shown in demonstrations, one needs to capture the user’s intent, rather than simply mimic the user, assuming user’s near-optimality. If we manage to infer the user’s intent as a function of their observed behavior, such a function can be *extrapolated* beyond the demonstrated behavior, potentially exceeding the performance of the demonstrator.

This idea underlies the T-REX (Trajectory-ranked Reward EXtrapolation) model suggested in Brown et al. (2019). The T-REX agent is provided with a number of ranked demonstrations. Ranks of trajectories convey information about preferences of the demonstrator. T-REX is flexible about the specific form of imposing ranking relations on demonstrated trajectories. One possibility is to rank all trajectories simultaneously based on some pre-specified measure of the quality of trajectories (e.g., it can be an “internal reward” of the demonstrator, i.e., some sort of the “ground-truth” cumulative reward⁹). Another possible specification avoids assigning a numerical value to the quality, but rather defines it as an ordinal preference relation defined on pairs of trajectories. As T-REX operates on pairs of (sub-) trajectories, as we will describe below, the second form of rank ordering based on pairwise comparison is sufficient for training of T-REX. On the other hand, clearly pairwise preference relations can be easily deduced once a quantitative absolute preference measure is provided for each individual trajectory, as in the first specification.

⁹An extension of the T-REX model called D-REX, to be presented next, allows one to proceed even when externally provided ranks are not available.

More formally, assume that we are given N ranked demonstrations τ_n that are ranked in the ascending order, so that demonstration τ_1 is the worst (has the lowest rank), and demonstration τ_N is the best (has the higher rank). The preference relations for a pair of trajectories τ_i, τ_j are written as $\tau_i \prec \tau_j$ if $i < j$.

The T-REX algorithm proceeds in two steps: (i) reward inference and (ii) policy optimization. Only the first step in this procedure amounts to a form of IRL that infers the (extrapolated) reward function from demonstrations. The second step amounts to using this extrapolated reward function with any particular algorithm for the direct RL in order to improve the performance by optimizing the RL policy. In what follows we focus on the first, IRL, step of the T-REX model.

The objective of the reward inference step of the T-REX model is to find a parameterized reward function $\hat{r}_\theta(s, a)$ which approximates the true reward function that the demonstrator is trying to optimize. This is formulated as a typical IRL objective, but the critical trick added by the T-REX model is that an additional structural constraint is imposed on the reward function. More specifically, cumulative rewards computed with this function should match the rank-ordering relation:

$$\sum_{(s,a) \in \tau_i} \hat{r}_\theta(s, a) < \sum_{(s,a) \in \tau_j} \hat{r}_\theta(s, a) \text{ if } \tau_i \prec \tau_j. \quad (11.129)$$

Let $\hat{J}_\theta(\tau_i) = \sum_t \gamma^t \hat{r}_\theta(s_t, a_t)$ be a discounted cumulative rewards on trajectory τ_i . We train T-REX by minimizing the following loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau_i, \tau_j \sim \Pi} \left[\xi \left(P \left(\hat{J}_\theta(\tau_i) < \hat{J}_\theta(\tau_j) \right), \tau_i \prec \tau_j \right) \right], \quad (11.130)$$

where Π is a distribution over pairs of demonstrations, and ξ is a binary loss function. The binary event probability P in Eq. (11.130) is modeled as a softmax distribution

$$P \left(\hat{J}_\theta(\tau_i) < \hat{J}_\theta(\tau_j) \right) = \frac{\exp \sum_{s,a \in \tau_j} \hat{r}_\theta(s, a)}{\exp \sum_{s,a \in \tau_i} \hat{r}_\theta(s, a) + \exp \sum_{s,a \in \tau_j} \hat{r}_\theta(s, a)}. \quad (11.131)$$

For the loss function $\xi(\cdot)$, a cross-entropy loss is used, so that the loss function becomes

$$\mathcal{L}(\theta) = - \sum_{\tau_i \prec \tau_j} \log \frac{\exp \sum_{s,a \in \tau_j} \hat{r}_\theta(s, a)}{\exp \sum_{s,a \in \tau_i} \hat{r}_\theta(s, a) + \exp \sum_{s,a \in \tau_j} \hat{r}_\theta(s, a)}. \quad (11.132)$$

This loss function trains a classifier to predict whether one trajectory is preferred over another one based on their realized total returns, and compares it with the preference label for this pair. The result is expressed in terms of a parameterized one-step reward function, as in other IRL approaches. In contrast to other methods,

T-REX finds rewards that are most consistent with *perceived goals* of the demonstrator, rather than with actual results of demonstrations. It is important to stress here that demonstrations themselves can be highly sub-optimal—what matters is that T-REX learns the *intent* of the demonstration, rather than simply assumes that they are already (nearly) optimal, as is done in, e.g., MaxEnt IRL. Once the intent is codified in a reward function, it can be extrapolated to other state-action combinations in the process of policy optimization. This can produce results exceeding the performance in demonstrations.

As was shown in Brown et al. (2019), T-REX produces better results than GAIL and other state-of-the-art adversarial and behavioral cloning methods on a number of tasks within the MuJoCo and Atari simulated environments. This implementation used a deep convolutional neural network (CNN) as a model of parameterized reward $\hat{r}_\theta(s, a)$. However, the T-REX approach is general, and can be applied using different architectures as well. In particular, it can be used for a finite MDP, which would require no function approximation at all. In the next section we will consider a simple financial example of T-REX for IRL with a discrete MDP.

8.4 D-REX: Disturbance-Based Reward EXtrapolation

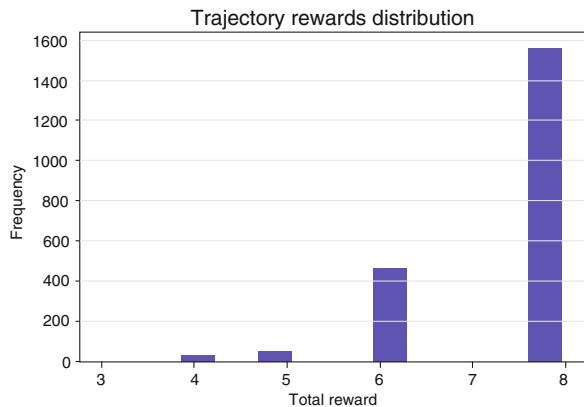
The T-REX algorithm that we have just presented appears simple and intuitively appealing, but it depends on the availability of rank labels for trajectory pairs. However, such ranking may not always be available. For example, a portfolio manager might have historical records of past trading portfolios but be in trouble providing pairwise ranking between them. The question is then what we can do with such scenarios.

A method that generalizes T-REX and provides automatically ranked demonstration is called D-REX (Disturbance-based Reward EXtrapolation) (Brown et al. 2019). The idea of this method is to use a ranking-based imitation learning method that injects different levels of noise into a policy learned using behavioral cloning. As was shown in Brown et al. (2019), this can be used to provide automated ranking to trajectories provided by a demonstrator, without any explicit ranking from the demonstrator. Once ranking labels are generated in this way, the algorithm follows the T-REX method.

9 Let Us Try It Out: IRL for Financial Cliff Walking

After some extensive theoretical introduction to inverse reinforcement learning and imitation learning given so far, we are now ready to apply IRL to a simple financial problem where results could be easily checked versus available ground-truth rewards.

Fig. 11.6 The distribution of cumulative rewards for simulated trajectories from the financial cliff walking (FCW) example



In this section, we apply three IRL algorithms (Max-Causal Entropy IRL, IRLF, and T-REX) with a discrete MDP, namely our financial cliff walking (FCW) example from Chap. 9, which we mentioned earlier as a possible simple test case for IRL.

Recall that the optimal policy for the FCW example is to deposit the minimum amount at the account at time $t = 0$, then do nothing until the very last step, at which point the account should be closed, with the reward of 10. We randomize this policy by adding a purely random component, and use it to produce sampled trajectories in the form of tuples (s_t, a_t, r_t, s_{t+1}) . The starting position is chosen to be $(1, 0)$ rather than $(0, 0)$ as in our RL example, which is chosen to avoid a deterministic move up at the first time step. Demonstrations with the total reward of 100 or less (i.e., trajectories leading to a bankruptcy) are marked as failed trajectories, and trajectories with a positive cumulative reward are marked as successful. Note that not all of them are optimal—the optimal strategy has the total reward of 10 and does not act until the very last step, only then withdrawing the entire amount. The distribution of rewards for successful trajectories is shown in Fig. 11.6.

We now present the results of the experiments, using the three IRL methods on this data, and compare them with the ground-truth rewards.

9.1 Max-Causal Entropy IRL

Recall that in MaxEnt IRL, all trajectories in demonstration are assumed to be nearly optimal, with some occasional deviations from optimality. The distribution of rewards in Fig. 11.6, obtained when only successful trajectories are given to the agent, appears to be a credible case for such an assumption, given that most of the trajectories have an optimal reward of 10 or a nearly optimal reward of 8.

The results of MaxEnt IRL are shown in Fig. 11.7. Evidently, MaxEnt IRL is able to capture the high reward of moving downwards from the state $(1, 10)$ (see the second graph on the right). The recovered reward for zero action is large as long as

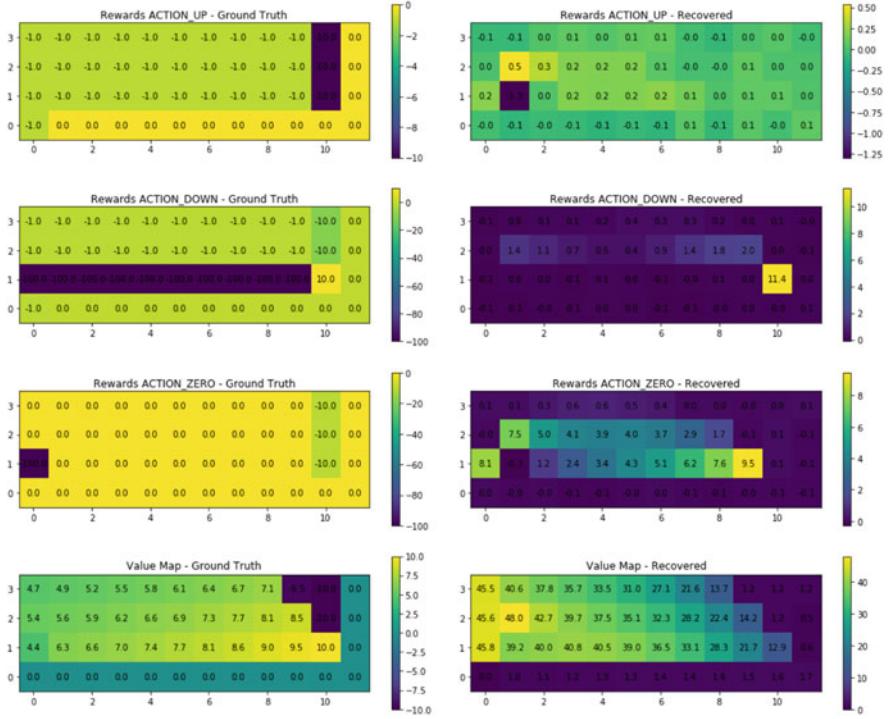


Fig. 11.7 Results of MaxEnt IRL on the FCW problem. (Left) The ground-truth rewards for all moves and the true value function. (Right) The recovered values

the agent is left with one unit of deposit until the last step. This does not correspond to the ground truth, however: The true reward over these steps is zero. MaxEnt IRL associates these state-action combinations with high-reward states simply because they are sub-parts of the optimal trajectories. Even though the true trajectory reward of 10 is obtained at the very last step, preceding steps on the optimal trajectory are perceived by the MaxEnt IRL as steps with high local reward. Another observation is that the MaxEnt IRL does not detect the cliff, while in fact avoiding the cliff is more important, in terms of the total reward, than any other actions. The reason is of course that MaxEnt IRL is oblivious to the cliff since it was only given successful trajectories.

9.2 IRL from Failure

Next we consider the IRLF (IRL from Failure) model. Here we use both sets of demonstrations with successful and unsuccessful trajectories. Both datasets have an equal number of observations. The results of IRLD are shown in Fig. 11.8.

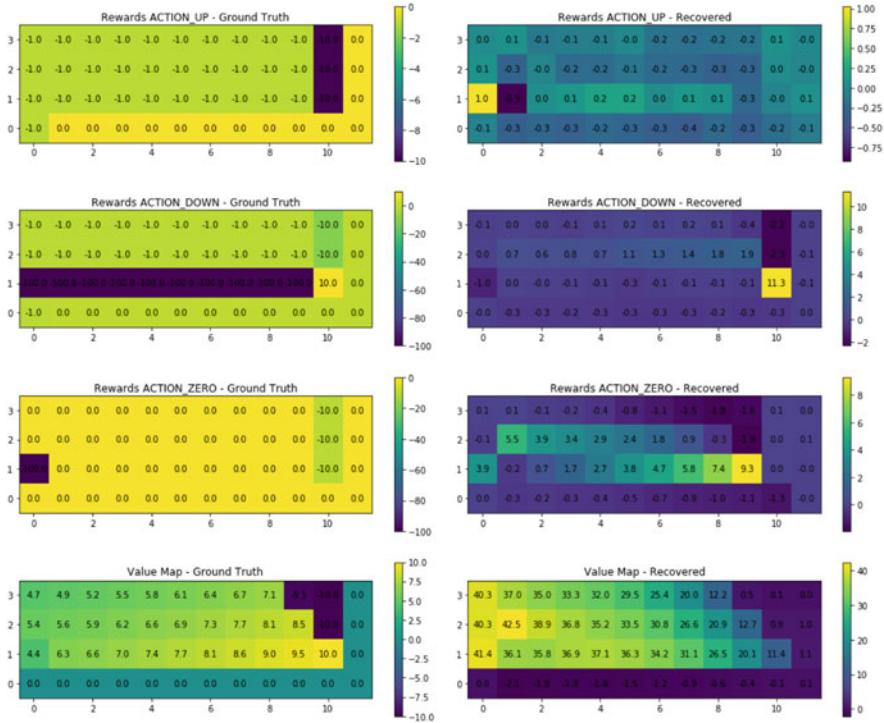


Fig. 11.8 Results of IRLF on the FCW problem. On the left: the ground-truth rewards for all moves and the true value function. On the right: the recovered values

IRLF shows a similar performance to MaxEnt IRL for our FCW problem. Though it is also given failed demonstrations, it does not produce a marked imprint of the strong negative reward from breaching the cliff. The recovered reward appears similar to the reward found by the MaxEnt IRL method.

9.3 T-REX

For T-REX, we provide the agent with an equal number of successful and failed trajectories which are not split into separate datasets, but rather given as elements of the same dataset. Each trajectory is ranked by the corresponding ground-truth rewards accumulated on it.

The results of T-REX are shown in Fig. 11.9. We can see that it performs better than the other two IRL methods. First, it differentiates between the negative reward of falling from the cliff from the first level from zero rewards obtained when already in the bankruptcy state. It also appears to be less error prone with assigning rewards for zero actions—instead of assigning positive rewards to all parts of trajectories

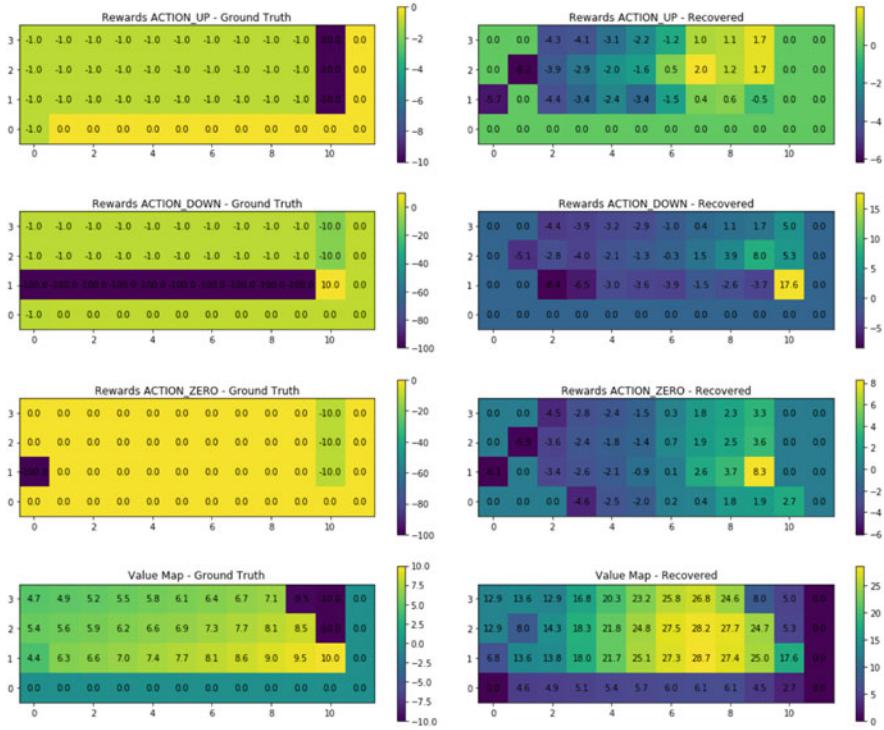


Fig. 11.9 Results of T-REX on the FCW problem. On the left: the ground-truth rewards for all moves and the true value function. On the right: the recovered values

with zero actions, it tends to assign negative rewards in the beginning and positive rewards at the end, which partially offset each other in terms of cumulative rewards.

9.4 Summary

As we have demonstrated with these simple examples, IRL is a challenging problem. Recall that IRL is an inverse problem of recovering a hidden signal (the reward, in our case) from noisy observations. Not every inverse problem enables a full recovery of the signal, some information is inevitably lost. Our experiment with a simple FCW environment confirms this. Because we operated in a discrete MDP setting, no function approximation was involved, enabling us to focus on the performance of the inference part itself. Our experiments demonstrate the reward ambiguity in IRL: the absolute values of recovered reward do not correspond to the absolute values of the ground-truth rewards. On the positive side, IRL methods, and especially the T-REX method, appear to capture the general structure of ground-truth rewards.

10 Financial Applications of IRL

In this section, we consider financial applications of IRL and IL. While to date there are only a handful of published research papers on applications of IRL in finance, we will consider a few interesting use cases from different areas of quantitative finance.

We will consider two types of IRL problems. In the first class, we consider inference of a reward function of a *particular* market agent. Such as agent can be either a human or a robot, i.e. a trading algorithm. We will discuss three use cases that belong in this class. The first one is the problem of identification of strategies in high-frequency futures trading. The second is the problem of learning the reward function of a risk-averse option trader. The third use case presents an IRL formulation for inference of the reward function of a portfolio investor.

The second type of IRL problems infers a “merged” reward function of *all* market agents, assuming that some market signals or even market dynamics are impacted by their collective actions. While this remains a single-agent IRL formulation, the agent here becomes an embodiment of some collective mode in the market, which can be informally interpreted as an “Invisible Hand” of the market. For this type of IRL problems, we again consider two use cases. In the first one, the collective action of all market participants is identified with the market investor sentiments (proxied by news sentiments). In the second use case, the collective action is taken to be the net inflow or outflow of capital in the given stock. As we will see below, such applications of IRL are capable of providing new insights into modeling on market dynamics.

10.1 Algorithmic Trading Strategy Identification

One of the first published financial applications of IRL was suggested by Yang et al. (2015). They addressed the problem of identification of high-frequency trading strategies (HFT) given observed trading histories. This problem is of particular interest to market operators and regulators seeking to prevent fraud and unfair trading practices. In addition, a systematic analysis of algorithmic trading practices helps regulators to produce regulations and policies that maintain the overall health of the market.

The identification of trading strategies is often addressed by practitioners using unsupervised learning techniques such as clustering of all algorithmic strategies. They use features obtained from cumulative statistics of activities on trading accounts. Clearly, such reliance on various cumulative statistics of trading activity appears a “blind” method that might be prone to identifying features that have little association with real objectives of strategies’ operators. Respectively, relying on such features to perform clustering of strategies might produce non-informative clusters with little explanatory power.

Inverse reinforcement learning offers an alternative to such a purely statistical (or data-mining) approach. If observed trading strategies are used to infer reward functions of traders (or algorithmic strategies), the learned rewards may provide a different set of more “intelligent” features—ones that are informative of perceived goals of financial agents. As was found in Yang et al. (2015), clustering based on a learned reward function provides better identified clusters of agents having similar reward functions.

Data collected by Yang et al. (2015) amounted to a month of order book audit trail data for E-Mini S&P 500 index traded on the Chicago Mercantile Exchange (CME) Globex electronic trading platform. The audit trail data includes all the order book events at a millisecond time resolution. Each record contains the following inputs: date, time, confirmation time (i.e., the time the order is confirmed by the order-matching engine), customer account, trader identification number, buy/sell flag, price, quantity, order type (market or limit), message type, and order ID.

State features are constructed using order volume misbalances between the best bid and the best ask prices, at three different levels of the limit order book. These three variables are further discretized into three levels (“high,” “medium,” “low”). In addition, the inventory level (holding position) is used as part of the state description. The action space is discretized in the following way: All limit and market orders are separately discretized into 10 buckets each. In addition, the market agent can cancel an existing limit order, which produces two more binary degrees of freedom. As a result, all possible actions are encoded in a binary vector of length 22. The total vector of state-action features is obtained by stacking together discretized values of states and actions.

Yang et al. (2015) use the Bayesian IRL approach of Ramachandran and Amirv (2007). In Bayesian IRL, the reward is assumed to be an unobserved random variable with a probability distribution $p(r)$. Given an observation O of a next state and action with transition probability $p(s', a|s)$, the posterior probability of a given value of the reward is obtained as

$$p(r|s, a, s') = \frac{p(r)p(s', a|r, s)}{\sum_r p(r)p(s', a|r, s)}. \quad (11.133)$$

Rewards related to a particular action a_m are modeled as Gaussian processes (GP) with zero mean and covariance matrix \mathbf{K}_m , leading to $r \sim N(0, \mathbf{K}_m)$. Rewards are therefore completely specified by the covariance matrix \mathbf{K}_m . Elements $k_m(s_i, s_j)$ of the covariance matrix are taken to be squared exponential kernels:

$$k_m(s_i, s_j) = e^{k_m \frac{1}{2}(s_i - s_j)^2} + \sigma_m^2 \delta_{s_i, s_j}. \quad (11.134)$$

The model parameter vector θ is therefore given by the values of k_m , σ_m , σ , where σ is a parameter controlling the noise level in the likelihood function. The GPIRL algorithm of Yang et al. (2015) iterates between two steps:

- For given values of parameters θ , maximize the posterior $p(r|O)$. This is equivalent to minimization of $-\log p(O|r) - \log p(r|\theta)$. The output of this

- step is the maximum a posteriori (MAP) value of the reward $r_{MAP}(\theta)$ which maximizes the value of the numerator in Eq. (11.133).
- Optimize parameters θ by maximizing $\log p(O|\theta, r_{MAP})$ which is obtained as the Laplace approximation of $p(\theta|O)$.

As shown in Yang et al. (2015), maximum posterior calculation in this GPIRL formulation can be reduced to convex optimization and thus can be done in a numerically efficient manner. Features obtained from learned reward functions were then used to perform clustering of algorithmic strategies. Clusters obtained in this way were found to be purer and more interpretable than clusters obtained by using features derived from cumulative account statistics.

10.2 Inverse Reinforcement Learning for Option Pricing

Inverse reinforcement learning can be used to infer the reward function of an option trader who sells a fixed number of European put options with maturity T and strike K on a stock whose value now (at time t) is S_t . To hedge exposure to fluctuations of stock price S_T at the option maturity, the option seller longs a portfolio made of a certain quantity of the stock and some cash. The option seller dynamically rebalances this hedge portfolio in order to mitigate risk of not meeting her obligation to sell the stocks at price K in scenarios when the price $S_T > K$. Rewards are defined as risk-adjusted returns on the option hedge portfolio.

We assume the same settings as we used for the QLBS model for option pricing in Chap. 10. Recall that the QLBS is a model of a small investor who does not move the market. This assumption is the same as in the classical Black–Scholes (BS) option pricing model. In contrast to the BS model, the QLBS model prices option by discrete-time sequential risk minimization. It shows that a simple Markowitz-like single-step (in time Δt) reward (negative risk) with risk aversion λ produces a semi-analytically tractable model that reduces, if the dynamics of the world are log-normal, to the standard BS model in the limit $\Delta t \rightarrow 0, \lambda \rightarrow 0$.

In Chap. 10, we considered a direct reinforcement learning problem with the QLBS model, where observed data includes states, actions, and rewards. Here we are concerned with the problem of learning both the reward function and the optimal policy assumed to be followed in the data. In other words, we take an IRL perspective of the option pricing problem.

While typically IRL is a harder problem than RL, and *both* are computationally hard, in the setting of the QLBS model both are about equally easy, due to a quadratic form of the reward function (10.35) and the absence of a feedback loop of traders' actions on the market.

Indeed, learning of the reward function amounts in this case to finding just one parameter λ using Eq. (10.35). To estimate this parameter, we adapt the setting of MaxEnt IRL, which is based on the general equations for G-learning (11.29):

$$\begin{aligned} F_t^\pi(\mathbf{s}_t) &= \mathbb{E}_{\mathbf{a}} \left[r(\mathbf{s}_t, \mathbf{a}_t) - \frac{1}{\beta} g^\pi(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}} [F_{t+1}^\pi(\mathbf{s}_{t+1})] \right] \\ \pi(\mathbf{a}_t | \mathbf{s}_t) &= \frac{1}{Z_t} \pi_0(\mathbf{a}_t | \mathbf{s}_t) e^{r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{t, \mathbf{a}} [F_{t+1}^\pi(\mathbf{s}_{t+1})]}. \end{aligned} \quad (11.135)$$

As we remarked in Sect. 3.3, Eq. (11.135) shows that one-step rewards $r(\mathbf{s}_t, \mathbf{a}_t)$ do *not* in general form an alternative specification of single-step action probabilities $\pi(\mathbf{a}_t | \mathbf{s}_t)$, which also require the value of $\gamma \mathbb{E}_{t, \mathbf{a}} [F_{t+1}^\pi(\mathbf{s}_{t+1})]$.

However, for a special case without market impact that we consider here, the analysis simplifies. Note that the last expression depends on the action a_t via the transition probabilities $p(s_{t+1}|s_t, a_t)$. Under the assumption that the agent is a small option trader whose actions do not impact market prices, we have $p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_t)$. In this case, the second term in the exponent in the policy formula, in the second of Eqs. (11.135), becomes independent of action a_t , and thus cancels out in the numerator and denominator of the policy equation. Therefore, for the special case without market impact (a feedback loop from action), the MaxEnt policy can be expressed solely in terms of one-step rewards:

$$\pi(a_t | X_t) = \frac{1}{Z} e^{r_t(X_t, a_t)}, \quad (11.136)$$

where Z is a normalization factor.

In the QLBS model, the expected one-step reward is (see Eq. (10.35))

$$r_t(X_t, a_t) := \mathbb{E}_t [R_t(X_t, a_t, X_{t+1})] = c_0(\lambda) + a_t c_1(\lambda) - \frac{1}{2} a_t^2 c_2(\lambda), \quad (11.137)$$

where, omitting for brevity the dependence on X_t , we defined

$$\begin{aligned} c_0(\lambda) &= -\lambda \gamma^2 \mathbb{E}_t [\hat{\Pi}_{t+1}^2], \quad c_1(\lambda) = \gamma \mathbb{E}_t [\Delta S_t + 2\lambda \gamma \Delta \hat{S}_t \hat{\Pi}_{t+1}], \\ c_2(\lambda) &= 2\lambda \gamma^2 \mathbb{E}_t [\left(\Delta \hat{S}_t \right)^2]. \end{aligned} \quad (11.138)$$

Combining this with the MaxEnt policy (11.136), we obtain

$$\pi(a_t | X_t) = \frac{1}{Z} e^{r_t(X_t, a_t)} = \sqrt{\frac{c_2(\lambda)}{2\pi}} \exp \left[-\frac{c_2(\lambda)}{2} \left(a_t - \frac{c_1(\lambda)}{c_2(\lambda)} \right)^2 \right]. \quad (11.139)$$

Thus, by combining an exponential distribution of the MaxEnt method with the quadratic expected reward (10.35), we ended up with a Gaussian action policy (11.139).¹⁰

Using Eq. (11.139), the log-likelihood of observing data $\{X_t^{(k)}, a_t^{(k)}\}_{k=1}^N$ is (omitting a constant factor $-\frac{1}{2} \log(2\pi)$ in the second expression)

$$LL(\lambda) = \log \prod_{k=1}^N p_\lambda \left(a_t^{(k)} \mid X_t^{(k)} \right) = \sum_{k=1}^N \left(\frac{1}{2} \log c_2^{(k)}(\lambda) - \frac{c_2^{(k)}(\lambda)}{2} \left(a_t^{(k)} - \frac{c_1^{(k)}(\lambda)}{c_2^{(k)}(\lambda)} \right)^2 \right), \quad (11.140)$$

where $c_i^{(k)}(\lambda)$ with $i = 1, 2$ stands for expressions (11.138) evaluated on the k -th path. As this is a concave function of λ , its unique maximum can be easily found numerically using standard optimization packages.

Note that optimization in Eq. (11.140) refers to one particular value of t . This calculation can be repeated independently for different times t , producing a curve $\lambda_{impl}(t)$ that could be viewed as a term structure of implied risk-aversion parameter.

To summarize this example, we see that when there is no market impact (a feedback loop in the system), MaxEnt IRL directly learns a one-step reward function. Parameters of this reward function can be estimated using the conventional maximum likelihood estimation with the MaxEnt stochastic policy.

10.3 IRL of a Portfolio Investor with G-Learning

In Chap. 10, we introduced G-learning with quadratic rewards and Gaussian time-varying policies (GTVP) as a tool to optimize a dynamic asset portfolio. Such a model-based approach to G-learning amounts to a probabilistic version of the well-known Linear Quadratic Regulator. Thus far, we have considered two formulations that correspond, respectively, to self-financing and non-self-financing portfolios. While the first formulation is appropriate for modeling activities of asset managers, the second formulation with cash-flows at intermediate times is appropriate for tasks of wealth management and financial planning.

Here we shall take the second formulation of G-learning for an individual agent such as a retirement plan contributor or an individual brokerage account manager, and consider its inverse formulation. This problem was set in Dixon and Halperin (2020) that presented two related algorithms called G-learner and GIRL (for G-learning IRL) to perform, respectively, the direct and inverse reinforcement learning

¹⁰Such stochastic action policy is clearly different from the greedy policy obtained with Q-learning used for direct reinforcement learning in the QLBS model. As MaxEnt IRL operates with stochastic policies, the corresponding “direct” reinforcement learning formulation for option pricing with stochastic policies would not be the QLBS model but rather its entropy-regularized version based on G-learning.

with G-learning for the retirement planning problem. In this section, we outline the GIRL algorithm from Dixon and Halperin (2020).

We assume that we are given a history of dollar-nominated asset positions in an investment portfolio, jointly with a portfolio manager's decisions that include both injections or withdrawals of cash from the portfolio and asset allocation decisions. Additionally, we are given historical values of asset prices and expected asset returns for all assets in the investor universe. As a concrete example, we can consider a portfolio of stocks and a single bond, but the same formalism can be applied to other types of assets.

Recall that in Sect. 6.3 we obtained the stochastic policy (see Eq. (10.172))

$$\pi(\mathbf{u}_t | \mathbf{x}_t) = \pi_0(\mathbf{u}_t | \mathbf{x}_t) e^{\beta(G_t^\pi(\mathbf{x}_t, \mathbf{u}_t) - F_t^\pi(\mathbf{x}_t))}. \quad (11.141)$$

The quadratic reward function considered in Sect. 6.3 corresponds to the quadratic action-value function (see Eqs. (10.170) and (10.165))

$$\begin{aligned} F_t^\pi(\mathbf{x}_t) &= \mathbf{x}_t^T \mathbf{F}_t^{(xx)} \mathbf{x}_t + \mathbf{x}_t^T \mathbf{F}_t^{(x)} + F_t^{(0)} \\ G_t^\pi(\mathbf{x}_t, \mathbf{u}_t) &= \mathbf{x}_t^T \mathbf{Q}_t^{(xx)} \mathbf{x}_t + \mathbf{x}_t^T \mathbf{Q}_t^{(xu)} \mathbf{u}_t + \mathbf{u}_t^T \mathbf{Q}_t^{(uu)} \mathbf{u}_t + \mathbf{x}_t^T \mathbf{Q}_t^{(x)} + \mathbf{u}_t^T \mathbf{Q}_t^{(u)} + Q_t^{(0)}, \end{aligned} \quad (11.142)$$

where

$$\begin{aligned} \mathbf{Q}_t^{(xx)} &= -\lambda \hat{\Sigma}_t + \gamma \left(\mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(xx)} \mathbf{A}_t + \tilde{\Sigma}_r \circ \bar{\mathbf{F}}_{t+1}^{(xx)} \right) \\ \mathbf{Q}_t^{(xu)} &= 2\mathbf{Q}_t^{(xx)} \\ \mathbf{Q}_t^{(uu)} &= \mathbf{Q}_t^{(xx)} - \boldsymbol{\Omega} \\ \mathbf{Q}_t^{(x)} &= 2\lambda \hat{P}_{t+1}(1 + \bar{\mathbf{r}}_t) + \gamma \mathbf{A}_t^T \bar{\mathbf{F}}_{t+1}^{(x)} \\ \mathbf{Q}_t^{(u)} &= \mathbf{Q}_t^{(x)} - \mathbb{1} \\ Q_t^{(0)} &= -\lambda \hat{P}_{t+1}^2 + \gamma F_{t+1}^{(0)}. \end{aligned} \quad (11.143)$$

Here \hat{P}_t is a target portfolio defined here as a mixture of a benchmark portfolio B_t (e.g. an index or a multiple of an index), and the current portfolio with the value $\mathbf{1}^T \mathbf{x}$ that grows at some fixed rate η , with ρ being the mixture coefficient.

$$\hat{P}_{t+1} = (1 - \rho)B_t + \rho\eta\mathbf{1}^T \mathbf{x} \quad (11.144)$$

Assume that we have historical data that includes a set of D trajectories ζ_i where $i = 1, \dots, D$ of state-action pairs $(\mathbf{x}_t, \mathbf{a}_t)$, where trajectory i starts at some time t_{0i} and runs until time T_i . Consider a single trajectory ζ from this collection, and set for this trajectory the start time $t = 0$ and the end time T . As individual trajectories are considered independent, they will enter additively in the final log-likelihood of the problem. We assume that dynamics are Markovian in the pair $(\mathbf{x}_t, \mathbf{u}_t)$, with a

generative model $p_\theta(\mathbf{x}_{t+1}, \mathbf{u}_t | \mathbf{x}_t) = \pi_\theta(\mathbf{u}_t | \mathbf{x}_t) p_\theta(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t)$, where Θ stands for a vector of model parameters.

The probability of observing trajectory ζ is given by the following expression:

$$P(\mathbf{x}, \mathbf{u} | \Theta) = p_\theta(\mathbf{x}_0) \prod_{t=0}^{T-1} \pi_\theta(\mathbf{u}_t | \mathbf{x}_t) p_\theta(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t). \quad (11.145)$$

Here $p(\mathbf{x}_0)$ is a marginal probability of \mathbf{x}_t at the start of the i -th demonstration. Assuming that the initial values \mathbf{x}_0 are fixed, this gives the following log-likelihood for data $\{\mathbf{x}_t, \mathbf{a}_t\}_{t=0}^T$ observed for trajectory ζ :

$$LL(\theta) := \log P(\mathbf{x}, \mathbf{u} | \Theta) = \sum_{t \in \zeta} (\log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t) + \log p_\theta(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t)). \quad (11.146)$$

Transition probabilities $p_\theta(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t)$ entering this expression can be obtained from the state equation

$$\mathbf{x}_{t+1} = \mathbf{A}_t (\mathbf{x}_t + \mathbf{u}_t) + (\mathbf{x}_t + \mathbf{u}_t) \circ \tilde{\boldsymbol{\varepsilon}}_t, \quad \mathbf{A}_t := \text{diag}(1 + \bar{\mathbf{r}}_t), \quad \tilde{\boldsymbol{\varepsilon}}_t := (0, \boldsymbol{\varepsilon}_t), \quad (11.147)$$

where $\boldsymbol{\varepsilon}_t$ is a Gaussian noise with covariance $\boldsymbol{\Sigma}_r$ (see Eq. (10.160)). Writing $\mathbf{x}_t = (x_t^{(0)}, \mathbf{x}_t^{(r)})$, where $x_t^{(0)}$ is the value of a bond position and $\mathbf{x}_t^{(r)}$ are the values of positions in risky assets, and similarly for \mathbf{u}_t and \mathbf{A}_t , this produces transition probabilities

$$p_\theta(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{a}_t) = \frac{e^{-\frac{1}{2} \boldsymbol{\Delta}_t^T \boldsymbol{\Sigma}_r^{-1} \boldsymbol{\Delta}_t}}{\sqrt{(2\pi)^N |\boldsymbol{\Sigma}_r|}} \delta(x_{t+1}^{(0)} - (1 + r_f)x_t^{(0)}), \quad \boldsymbol{\Delta}_t := \frac{\mathbf{x}_{t+1}^{(r)}}{\mathbf{x}_t^{(r)} + \mathbf{u}_t^{(r)}} - \mathbf{A}_t^{(r)}, \quad (11.148)$$

where the factor $\delta(x_{t+1}^{(0)} - (1 + r_f)x_t^{(0)})$ captures the deterministic dynamics of the bond part of the portfolio. As this term does not depend on model parameters, we can drop it from the log-transition probability, along with a constant term $\sim \log(2\pi)$. This produces

$$\log p_\theta(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{a}_t) = -\frac{1}{2} \log |\boldsymbol{\Sigma}_r| - \frac{1}{2} \boldsymbol{\Delta}_t^T \boldsymbol{\Sigma}_r^{-1} \boldsymbol{\Delta}_t. \quad (11.149)$$

Substituting Eqs. (11.141), (11.143), (11.149) into the trajectory log-likelihood (11.146), we put it in the following form:

$$LL(\theta) = \sum_{t \in \zeta} \left(\beta (G_t^\pi(\mathbf{x}_t, \mathbf{u}_t) - F_t^\pi(\mathbf{x}_t)) - \frac{1}{2} \log |\boldsymbol{\Sigma}_r| - \frac{1}{2} \boldsymbol{\Delta}_t^T \boldsymbol{\Sigma}_r^{-1} \boldsymbol{\Delta}_t \right), \quad (11.150)$$

where $G_t^\pi(\mathbf{x}_t, \mathbf{u}_t)$ and $F_t^\pi(\mathbf{x}_t)$ are defined by Eqs. (11.143).

Table 11.1 The G-learning agent parameters used for portfolio allocation together with the values estimated by GIRL

Parameter	G-learner	GIRL
ρ	0.4	0.406
λ	0.001	0.000987
η	1.01	1.0912
ω	0.15	0.149

The log-likelihood (11.150) is a function of model parameter vector $\theta = (\lambda, \eta, \rho, \Omega, \Sigma_r, \Sigma_p, \bar{\mathbf{u}}_t, \bar{\mathbf{v}}_t)$ (recall that β is a regularization hyper-parameter which should not be optimized in-sample). We can simplify the problem by setting $\bar{\mathbf{v}}_t = 0$ and $\bar{\mathbf{u}}_t = \bar{\mathbf{u}}$ (i.e. take a constant mean in the prior). In this case, the vector of model parameter to learn with IRL inference is $\theta = (\lambda, \eta, \rho, \Omega, \Sigma_r, \Sigma_p, \bar{\mathbf{u}})$. A “proper” IRL setting would correspond to only learning parameters of the reward function $(\lambda, \eta, \rho, \Omega)$ while keeping parameters $(\Sigma_r, \Sigma_p, \bar{\mathbf{u}})$ fixed (i.e. estimated outside of the IRL model). The log-likelihood defined in Eq.(11.150) is concave and has a unique maximum, or equivalently its negative is convex and has a unique minimum.

The GIRL algorithm thus amounts to convex optimization which can be performed very efficiently using standard optimization software. The performance of the algorithm was evaluated in Dixon and Halperin (2020) using a simulated environment where an “alpha model” for expected returns needed as inputs to the algorithm is designed to have a weak predictive power, as expected from the performance of equity returns predictive models in the real life, see Fig. 11.11.

Experiments performed in Dixon and Halperin (2020) used a G-learning agent with arbitrarily chosen reward parameters to generate sequences of nearly optimal actions (cash inflows and portfolio rebalancing) that maximize the total expected reward in the simulated environment. It was found that the GIRL algorithm is able to accurately infer the correct reward parameters from the demonstrated behavior, as illustrated in Table 11.1.

The G-learner takes as input the expected risky asset returns \bar{r}_t together with the covariance of the risk asset return, Σ_r . As shown in Figure 11.10, even using arbitrary reward parameters chosen in Table 11.1 results in superior Sharpe ratios when compared with an equally weighted portfolio that is never rebalanced over the investment horizon. The G-learner uses the alpha-model to consistently produce superior returns in a multi-period setting using a locally quadratic reward function. The G-learner trains in a few seconds on a portfolio of 100 assets on standard hardware.

GIRL imitates the G-learner by minimizing a loss function over the state-action trajectories generated by the G-learner. The GIRL learned parameters in Table 11.1 are observed to be close to the G-learner parameters up to sampling error and numerical accuracy. Consequently GIRL is observed to imitate the G-learner — the sample averaged portfolio returns closely track each other in Figure 11.10. The error in the learned G-learner parameters results in a marginal decrease in the Sharpe ratio, as reported in the parentheses of the legend in Figure 11.10.

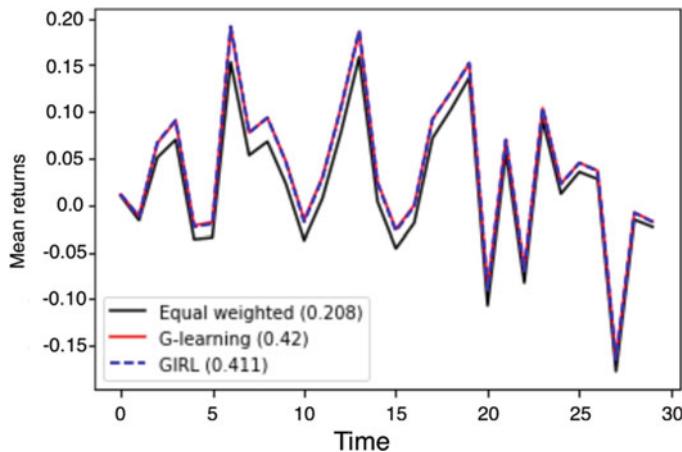


Fig. 11.10 The sample mean portfolio returns are shown over a 30 quarterly period horizon (7.5 years). The black line shows the sample mean returns for an equally weighted portfolio without rebalancing. The red line shows a G-learning agent, for the parameter values given in Table 11.1. GIRL imitates the G-learning agent and generates returns shown by the blue dashed line. Sharpe ratios are shown in parentheses

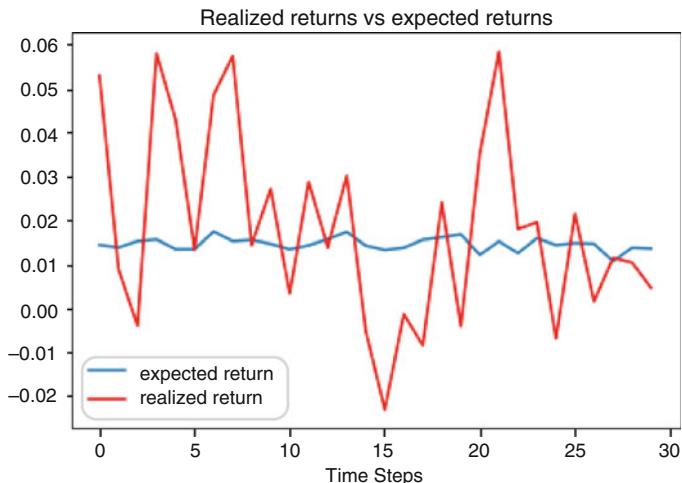


Fig. 11.11 The sample mean realized returns are plotted against the sample mean expected returns and observed to be weakly correlated

The two algorithms, G-Learner and GIRL, can be used either separately or in a combination. In particular, their combination could be used in robo-advising by modeling the actual human agents as G-learners, and then use GIRL to infer the latent objectives (rewards) of these G-learners. GIRL would then be able to imitate the best human investors, and thus could be offered as a robo-advising service to clients that would allow them to perform on par with best performers among all investors.

10.4 IRL and Reward Learning for Sentiment-Based Trading Strategies

Market prices of tradable securities such as stocks or bonds are significantly impacted by trading activities and resulting volume misbalances. Consequently, trading decisions of market participants are driven by both common factors predictive of future asset performance (those that are shared by all market players in their decision-making), and idiosyncratic decision-making factors that might be more tuned to specific objectives or strategies of market operators.

One of the common factors commonly accounted for by a majority of market participants is investor sentiment. Various proxies to such investor sentiment were considered in the financial literature. One popular approach is to use news sentiment scores computed by companies such as Bloomberg, Thomson-Reuters, or RavenPack, as measures of investor sentiment that expresses a general public mood towards a given stock.

One way to explore a link between a proxy to investor sentiment and future market movements is to use regressions where investor sentiments are used as inputs, and stock returns serve as outputs. This is a supervised learning approach. It does not try to view market sentiment as the result of some decision-making, but rather treats them as given and fixed inputs to the prediction problem.

Another approach is to treat investor sentiments as *actions* of market players observed in different states of the market. This sets up an MDP formulation for the problem. The advantage of this approach is that market sentiments might be adaptive to the state of the market, in the same way as a policy in an MDP problem is a function of a state. The MDP formulation therefore captures co-dependencies between sentiments and market states.

As rewards corresponding to actions defined by investor sentiments are not observable, we deal here with learning without rewards, which is the setting of IRL. Such an IRL view on learning the link between investor sentiments and market price dynamics was suggested by Yang et al. (2018). The objective of such modeling is to find *useful high-level features* that can be used to construct better predictive models for equity returns. In the IRL approach, these features are constructed by assuming that observed investor sentiments (actions) are maximizing unknown cumulative rewards over observed trajectories consisting of states and actions.

For a practical implementation, the authors of Yang et al. (2018) used news sentiments from Thomson-Reuters news analytics as a proxy of investor sentiments towards the US equity market. Sentiment scores for all stocks in the S&P 500 index (SPX) are aggregated into a common market-wide sentiment, which is considered a “collective action” (or voting) by all market agents. This action space is further discretized in three states corresponding, respectively, to high, medium, and low sentiments. To learn the reward function, Yang et al. (2018) use a probabilistic GPIRL approach, similar to one used in Yang et al. (2015). Experiments have shown that an adaptive trading system based on learned reward functions performed better than benchmark strategies based on supervised learning approaches.

We note that the approach of Yang et al. (2018) is intrinsically interesting as an example of dimension reduction in an MDP problem. Indeed, an investor sentiment is by construction a metric for the collective response of *all* market participants simultaneously. Therefore, if we identify investor sentiment with actions of an agent, such an agent should simultaneously embody *all* market participants. Even though it deals with market dynamics that include many interacting financial agents, the problem is formulated here as a single-agent IRL problem. This considerably simplifies analysis. We will return to the idea of modeling market dynamics with a single agent in the next section.

10.5 IRL and the “Invisible Hand” Inference

Investor sentiments, albeit collectively representing all market participants, are nevertheless not directly related to actual actions of market players—the latter amount to *trading* in different securities, rather than holding views on their future values. We can arrive at another single-agent model that relates to the market as a whole, if we start with a multi-agent view of a market. In this approach, agents may invest in all stocks from a market portfolio (e.g., the SPX portfolio of the S&P 500 stocks). At each time step, different agents may buy or sell different stocks, or simply hold their trading books.

In general, the market is filled with heterogeneous agents in various forms. There are rational agents who maximize a certain utility function. Then there are agents with bounded rationality, which is a term we will discuss in more details below. There may even be totally irrational agents—“noisy traders.” At each given time step, they would in general take different actions, as determined by their perceived goals.

On the other hand, market dynamics are driven, at each step, by a combination of trading signals, actions of all market agents, and an “innovation noise” resulting from new information that becomes available to market agents. The SPX portfolio obtained with these dynamics is a *market-optimal* portfolio which often serves as a benchmark portfolios for active management funds that try to “beat the market.”

We may think of the dynamics of this market-optimal portfolio as stochastic market dynamics partially affected by actions of *one* single agent. Such an agent

could be identified with a dynamic “collective mode,” or an “Invisible Hand” of the market. In other words, this agent acts like the sum, rather than the average of all agents in the market. Such an approach to modeling market dynamics was suggested in Halperin and Feldshteyn (2018). Our exposition in this section provides a brief overview of the IRL approach of Halperin and Feldshteyn (2018).

The objective of IRL for a market-wide agent is to find its reward function from observed market dynamics. To gain insights into possible parametric specifications for the reward function, it is instructive to first consider a simple one-period portfolio optimization problem. In this case, the expected reward coincides with the action-value function. Therefore, for this setting, finding an optimal investment policy is equivalent to maximizing the expected reward (or expected utility) of the investment.

Assume for the moment that the agent is rational¹¹ and has a certain utility $U(\mathbf{a})$, where \mathbf{a} is a vector of asset allocations, the direct portfolio optimization problem amounts to maximizing $U(\mathbf{a})$ given all its inputs, and subject to all constraints on asset allocations \mathbf{a} .

Instead of solving this direct optimal control problem, we can consider its *inverse*. Indeed, the market-optimal portfolio is already known from the market itself as market-optimal allocations are given by total market capitalizations of all stocks in the market index. We could use this information to find the utility from the optimal solution. This corresponds to an “inverse problem” of portfolio optimization.

This is exactly the approach of the celebrated Black–Litterman (BL) model of optimal asset allocation (Black and Litterman 1991). The BL model inverts the single-period (non-dynamic) Markowitz optimal portfolio theory. The Markowitz model computes the optimal portfolio a given investors’ views of expected returns $\bar{\mathbf{r}}$ and future covariances Σ of stock returns, by maximizing the utility function $U(\mathbf{a}) = \mathbf{a}^T \bar{\mathbf{r}} - \lambda \mathbf{a}^T \Sigma \mathbf{a}$, where λ is a risk-aversion parameter. Consequently, expected returns $\bar{\mathbf{r}}$ depend on values of predictive signals \mathbf{z} .

Flipping the problem on its head, the BL model takes as an input the market-optimal portfolio, assuming that it is obtained by maximization of the Markowitz utility, and infers from it market-implied views of expected returns. The latter translate into market-implied views of common predictive signals \mathbf{z} . Along with learning market-implied views, the BL model enables the user to infer the potential impact of investors *private* signals on the performance of the investment portfolio.

The BL model was explicitly reformulated as an *inverse optimization* problem by Bertsimas et al. (2012). While this provides an inverse optimization perspective of the BL approach, it remains a non-dynamic (one-period) formulation. This might be problematic if we are interested in signals whose dynamics extends over a few (or many) trading periods. Handling such effects requires a multi-period (dynamic) model of market returns. As IRL is based on learning from state-action sequences,

¹¹See below for more on the rationality assumption.

it provides a convenient framework for a dynamic, agent-based extension of the inverse optimization BL approach of Bertsimas et al. (2012).

Once we set on a single-agent formulation in the IRL approach to modeling market dynamics, it is important to clarify an “amount of rationality” of this agent. Traditional models of IRL or inverse stochastic control that are based on a concept of fully rational agents may *not* be well suitable for such IRL problem. Indeed, as market prices are impacted by actions of many individual investors, each potentially pursuing different investment objectives, an agent obtained by a direct summation of such real agents may not have a particular reward (utility) function that would correspond to one fixed type of a rational behavior.

A better assumption would be to have a collective agent that is *not* fully rational, but rather is only *bounded-rational*. A bounded-rational agent does not necessarily act to increase its total cumulative rewards on the action-value function. Instead, its objective function is made by a given parametric reward function penalized by an information cost (11.23) of updating policy π from some reference policy π_0 . The relative weight between the reward function and the information cost is controlled by an “inverse-temperature” parameter β , such that the limit $\beta \rightarrow \infty$ corresponds to a fully rational behavior, while the limit $\beta \rightarrow 0$ describes a maximally irrational behavior that sticks to a prior belief π_0 and is not adaptive to changes in processed information. Mathematically, optimization of a bounded-rational agent is equivalent to G-learning discussed in Sect. 3.3, where now the information cost is interpreted as a rationality penalty for updating the policy.

> Bounded Rationality

The concept of bounded rationality of financial agents was first suggested by Simon (1956) when he proposed to go beyond the more traditional rational investor models to better explain the behavior of real-world agents. The latter models follow the Von Neumann–Morgenstern expected utility approach in assuming that market agents are fully rational, and each of them maximizes a well-defined utility (reward) function. If we further assume that all agents have the same utility function, we obtain a model where a market-optimal portfolio is controlled by one fully rational representative agent. In particular, in the capital asset pricing model (CAPM), the utility function of such an agent is the Markowitz quadratic utility. But the assumption of perfect homogeneity of market agents might be too idealistic, as real financial markets are often populated with very heterogeneous types of investors. A formal approach to quantifying deviations from a perfectly rational behavior is therefore desired for a single-agent modeling of market dynamics.

(continued)

A convenient computational framework for modeling bounded-rational agents was proposed by Ortega and Braun (2013) who suggested to use information costs (11.23) as a quantitative measure of differences from a purely rational behavior. As was further discussed by Ortega et al. (2015), this metric computes the number of bits an agent requires to update its policy from a prior pre-specified policy π_0 .

Clearly, once we define a market-wide agent, its actions are not directly observable because it represents the market as a whole. Such agent has no trading counterparty—it trades with itself, which can be mathematically described as self-learning by a self-play, as driven by its reward function.¹² The agent’s actions amount to maintaining a partial control of the market portfolio.

Though actions of such a market-wide agent are not directly observable, they impact market prices via a price impact mechanism. Heating or cooling of the market via a price impact is the only observable effect of actions of the “Invisible Hand” agent. We assume a simple linear impact model for the impact of agent’s actions on the environment (the market). The return on a stock is therefore modeled as a linear function of the action, which is additionally contaminated by a noise term that approximates an aggregated “non-coherent” part of trading decisions of all market agents. In the simplest case, the noise can be modeled as a Gaussian white noise; however, including more complex volatility dynamics is possible as well.

Under these assumptions, the dynamic equation for the total market capitalization \mathbf{x}_t of all stocks with agent’s controls \mathbf{u}_t is the same as those obtained in Eq. (10.123) in Sect. 5.14 in Chap. 10, which we repeat here for convenience¹³:

$$\begin{aligned}\mathbf{x}_{t+1} &= (1 + \mathbf{r}_t) \circ (\mathbf{x}_t + \mathbf{u}_t) \\ &= \left(1 + r_f + \mathbf{Wz}_t - \mathbf{M}^T \mathbf{u}_t + \varepsilon_t\right) \circ (\mathbf{x}_t + \mathbf{u}_t) \\ &= (1 + r_f)(\mathbf{x}_t + \mathbf{u}_t) + \text{diag}(\mathbf{Wz}_t - \mathbf{Mu}_t)(\mathbf{x}_t + \mathbf{u}_t) + \varepsilon(\mathbf{x}_t, \mathbf{u}_t).\end{aligned}\quad (11.151)$$

¹²Recall in this context the example in Sect. 5.13 in Chap. 10 that showed that a single-agent learning in the setting of G-learning can be equivalently formulated as a solution for a Nash equilibrium in a zero-sum two-agent game.

¹³Our presentation in Sect. 5.14 in Chap. 10 refers to a general investment portfolio, where \mathbf{x}_t stands for a vector of dollar-valued positions in all stocks, and \mathbf{u}_t are adjustments of these positions. Here we apply this formalism to the whole market portfolio, where \mathbf{x}_t becomes the vector of total market capitalizations of all stocks in the market.

Here \circ stands for an element-wise (Hadamard) product, \mathbf{z}_t is the vector of predictive signals, \mathbf{W} are coefficients, and M is a matrix of market impacts that is assumed here to be diagonal with elements μ_i , $\mathbf{M} := \text{diag}(\mu_i)$, and

$$\varepsilon(\mathbf{x}_t, \mathbf{u}_t) := \varepsilon_t \circ (\mathbf{x}_t + \mathbf{u}_t) \quad (11.152)$$

is the multiplicative noise term. Equation (11.151) shows that the dynamics are non-linear in controls \mathbf{u}_t due to the market impacts $\sim \mu_i$. More specifically, when friction parameters $\mu_i > 0$, the state equation is linear in \mathbf{x}_t , but quadratic in controls \mathbf{u}_t . In the limit when all $\mu_i \rightarrow 0$, the dynamics become linear.

For both cases of linear and non-linear dynamics, the model of a bounded-rational market-wide agent is mathematically equivalent to a time-invariant version of G-learning presented in Sect. 5.13 in Chap. 10. The reason is that the same KL regularization used in G-learning is interpreted as an information cost of a bounded-rational agent to update from a prior policy, and is therefore used to control the amount of rationality of the agent, as was suggested in a different setting by Ortega and Braun (2013), Ortega et al. (2015).

When dynamics are linear while rewards are quadratic, dynamic portfolio optimization amounts to a probabilistic version of the Linear Quadratic Regulator (LQR). We have described the solution for this case in Sect. 5.15 in Chap. 10. As we found there, in the zero-friction limit the optimal policy is a Gaussian policy $\pi(\mathbf{u}_t | \mathbf{x}_t)$ whose mean is linear in the state \mathbf{x}_t :

$$\pi(\mathbf{u}_t | \mathbf{x}_t) = \frac{1}{\sqrt{(2\pi)^n |\tilde{\Sigma}_p|}} e^{-\frac{1}{2}(\mathbf{u}_t - \tilde{\mathbf{u}}_t - \tilde{\mathbf{v}}_t \mathbf{x}_t)^T \tilde{\Sigma}_p^{-1} (\mathbf{u}_t - \tilde{\mathbf{u}}_t - \tilde{\mathbf{v}}_t \mathbf{x}_t)}, \quad (11.153)$$

where $\tilde{\mathbf{u}}_t$, $\tilde{\mathbf{v}}_t$, and $\tilde{\Sigma}_p$ are parameters defined in Sect. 5.15 in Chap. 10.

When dynamics are non-linear, i.e. market impacts $\mu_i > 0$, analysis becomes considerably more complicated. Furthermore, when IRL is applied for a single-agent model of the market as a whole, actions of the agent become unobservable. One possible approach to address non-linearity is to set up a computational scheme that iterates between steps of tuning policy parameters and linearization of dynamics equations, all while treating the agent's actions as unobserved variables.

Such a computational method was developed in Halperin and Feldshteyn (2018) using a variational EM algorithm. The most important implication from the analysis of this algorithm is that even in the presence of market frictions $\mu_i > 0$, and absent of additional constraints, the optimal policy still has the same Gaussian form as in Eq. (11.153), with a mean that is linear in the state \mathbf{x}_t , and parameters that can be computed from the variational EM algorithm.

To obtain more constrained dynamics, we take a deterministic limit for this policy for a special case of a zero intercept. This produces a linear deterministic policy

$$\mathbf{u}_t = \Phi_t \mathbf{x}_t, \quad (11.154)$$

where Φ_t is a matrix of parameters that can be computed from the original model parameters. In general, this parameter is time-dependent as it can depend linearly on predictive signals \mathbf{z}_t . Note that the reason that Eq. (11.154) does not have an intercept is that the agent should not invest in stocks with a strictly zero value.

The linear action policy (11.154) describes the optimal action of the market-wide agent that maximized a specific quadratic (Markowitz) reward penalized by the KL information cost term, as was presented in Sect. 5.13 in Chap. 10. Alternatively, we could use G-learning with a different reward function.

Importantly, for different cost-adjusted reward functions, the resulting optimal action policy would be different from a linear one. In a general case, a deterministic policy in an MDP is a deterministic function of the state. We can write it as $\mathbf{u}_t = \Phi(\mathbf{x}_t, \mathbf{z}_t)$, where $\Phi(\mathbf{x}_t, \mathbf{z}_t)$ is a differentiable function that satisfies $\Phi(0, \mathbf{z}_t) = 0$. Their functional form is determined by a particular reward function chosen for the problem. In this case, Eq. (11.154) can be thought of as a leading-order Taylor approximation that uses the general Taylor expansion of a non-linear function $\Phi(\mathbf{x}_t, \mathbf{z}_t)$:

$$\mathbf{u}_t = \Phi(\mathbf{x}_t, \mathbf{z}_t) = \frac{\partial \Phi(\mathbf{x}_t, \mathbf{z}_t)}{\partial \mathbf{x}_t} \Big|_{\mathbf{x}_t=0} \mathbf{x}_t + \frac{1}{2} \frac{\partial^2 \Phi(\mathbf{x}_t, \mathbf{z}_t)}{\partial \mathbf{x}_t^2} \Big|_{\mathbf{x}_t=0} \mathbf{x}_t^2 + \dots \quad (11.155)$$

Assuming a deterministic policy such as (11.154) or (11.155), we may question its implications for the dynamics. To this end, we note that once we establish the policy (11.154) as a deterministic function of the state variable \mathbf{x}_t , we can simply plug this expression into Eq. (11.151) to give a dynamic equation that does not contain the control variable \mathbf{u}_t .

This produces drastically different results, depending on whether friction parameters are non-zero or ignored. Let us consider first the case of a linear action policy (11.154). Substituting Eq. (11.154) into Eq. (11.151) and simplifying, we obtain

$$\Delta \mathbf{x}_t = \mu \circ \phi \circ (1 + \phi) \circ \mathbf{x}_t \circ \left(\frac{\phi + (1 + \phi)(r_f + \mathbf{w}\mathbf{z}_t)}{\mu\phi(1 + \phi)} - \mathbf{x}_t \right) + (1 + \phi) \circ \mathbf{x}_t \circ \varepsilon_t^{(r)}. \quad (11.156)$$

Introducing parameters

$$\kappa \Delta t = \mu \circ \phi \circ (1 + \phi), \quad \theta(\mathbf{z}_t) = \frac{\phi + (1 + \phi)(r_f + \mathbf{w}\mathbf{z}_t)}{\mu\phi(1 + \phi)}, \quad \sigma(\mathbf{x}_t) \sqrt{\Delta t} = (1 + \phi) \circ \mathbf{x}_t \quad (11.157)$$

(here Δt is a time step) and replacing $\varepsilon_t^{(r)} \rightarrow \varepsilon_t$, we can write Eq. (11.156) more suggestively as

$$\Delta \mathbf{x}_t = \kappa \circ \mathbf{x}_t \circ (\theta(\mathbf{z}_t) - \mathbf{x}_t) \Delta t + \sigma(\mathbf{x}_t) \sqrt{\Delta t} \circ \varepsilon_t. \quad (11.158)$$

Note that this equation has a *quadratic* drift term. It is quite different from models with a *linear* drift such as the Ornstein–Uhlenbeck (OU) process. In the limit $\mu \rightarrow$

$0, \phi \rightarrow 0$, Eq. (11.158) reduces to the log-normal return model given by Eq. (10.80) without the action term \mathbf{u}_t :

$$\frac{\Delta \mathbf{x}_t}{\mathbf{x}_t} = r_f + \mathbf{w}\mathbf{z}_t + \varepsilon_t. \quad (11.159)$$

Therefore, the conventional log-normal return dynamics (with signals) is reproduced in our framework in the limit $\mu \rightarrow 0, \phi \rightarrow 0$. However, when parameters μ, ϕ are small but non-zero, Eqs. (11.159) and (11.158) describe *qualitatively* different dynamics.

In particular, while Eq. (11.159) is scale invariant with respect to scale transformations $\mathbf{x}_t \rightarrow \alpha \mathbf{x}_t$ with α being a scaling parameter, the non-linear mean-reverting dynamics (11.158) are *not* scale invariant. This is of course due to the fact that our market-wide agent aggregates all agents in the market. As their individual trade impacts induce a dependence of dynamics on a dimensional market impact parameter μ , scale invariance is broken in the resulting market dynamics (11.158).

Therefore, even if parameters κ, ϕ are small but non-vanishing, Eq. (11.158) produces a potentially complex non-linear dynamics with broken scale invariance and ensuing multi-period autocorrelations. These non-linear dynamics with a *dynamically* generated mean reversion level $\theta(\mathbf{z}_t)$ are produced from simple linear dynamics (11.151) with a linear control \mathbf{u}_t . Both the *level* and the *speed* of mean reversion have very clear origins: as can be seen from Eqs. (11.158), the level $\theta(\mathbf{z}_t)$ is driven by external signals \mathbf{z}_t , which makes intuitive sense. On the other hand, the *speed* of reverting to such a “target” price is proportional to the market impact parameter vector μ and is thus also intuitive.

We note that both the dynamically generated mean reversion which leads to long-term correlations and a dynamic adaptation of an optimal agent’s actions to external signals \mathbf{z}_t are phenomena that are typical for self-organizing systems, see, e.g., Yukalov and Sornette (2014). Therefore, the model of self-learning by a fictitious self-playing agent, which imitates simultaneously all traders in the market proposed in Halperin and Feldshteyn (2018), provides a specific illustration of equivalence between self-organization and decision-making that was suggested in Yukalov and Sornette (2014).

In a one-dimensional (1D) case with a constant mean reversion level $\theta(\mathbf{z}_t) = \theta$, Eq. (11.158) produces the following dynamics for a rescaled variable $s_t = x_t/\theta$:

$$\Delta s_t = \mu s_t (1 - s_t) + \sigma \sqrt{\Delta t} s_t \varepsilon_t, \quad \mu := \kappa \theta \Delta t. \quad (11.160)$$

Dynamics described by Eq. (11.160) or its noiseless limit $\sigma \rightarrow 0$ are widely encountered or used in physics and biology. In particular, the limit $\sigma \rightarrow 0$ of Eq. (11.160) describes the logistic map dynamics that arises, e.g., in the Malthus–Verhulst model of population growth (see, e.g., Kampen (1981)), or in Feigenbaum bifurcations in the logistic map chaos that arise when $3 \leq \mu < 4$ in Eq. (11.160), see, e.g., Sternberg (2010). When $\sigma > 0$, Eq. (11.160) describes a logistic map with a multiplicative thermal noise, which may produce highly complex dynamics (Baldovin and Robledo 2005).

We can also consider a continuous-time limit of 1D dynamics implied by Eq.(11.158):

$$dx_t = \kappa x_t (\theta - x_t) dt + \sigma x_t dW_t, \quad (11.161)$$

where W_t is a standard Brownian motion. This 1D process is known in the economics and finance literature as a Geometric Mean Reversion (GMR) process. The GMR model (11.161) was used by Dixit and Pindyck (1994), and its properties were further studied by Ewald and Yang (2007) who have shown that this process is bounded, non-negative, and has a stationary distribution under the constraint $2\kappa\theta > \sigma^2$. Rather than postulating such mean-reverting dynamics, the model presented in Halperin and Feldshteyn (2018) *derives* them (in a multivariate setting) from an underlying dynamic optimization problem of a bounded-rational agent.

To summarize, the IRL approach to modeling the market dynamics as a partially controlled MDP model with a linear optimal control produces, upon converting the problem into an open loop control formulation, the non-stationary multivariate GMR process (11.158) (or, in a one-dimensional setting, its 1D counterpart Eq.(11.161)). This process has a quadratic drift, in contrast to the GBM or OU models which have linear drifts. When applied to the market as a whole, IRL is capable of producing dynamic market models with a non-linear drift.

An interesting question is how the resulting market dynamics equations would change if we kept both the linear and quadratic term in the general form (11.155) of the optimal policy function. Evidently, retaining this term would change the drift in Eq.(11.158) from quadratic to *cubic*. As we will discuss in the next chapter, such a modification of the resulting dynamic market model might be desirable in order to extend the model to describe various market regimes, including in particular a regime when the effective mean reversion rate $\kappa < 0$.

11 Summary

In this chapter, we presented the concepts and methods of inverse reinforcement learning (IRL) and imitation learning (IL). As the whole field of IRL and IL is currently being actively pursued (outside of finance) by many researchers in ML, this chapter's goal was to provide a sufficiently high-level review to help the reader navigate the research literature and follow new developments. In particular, we provided a review of adversarial approaches to IRL and IL that have been recently trending in the ML literature.

As we mentioned in the introduction to this section, while the whole field of IRL and IL continues to generate significant interest and new ideas among researchers applying these methods in robotics and video games, current financial applications are still scarce. Financial applications put stringent constraints on IRL or IL methods to be applicable in finance: they need to operate with continuous (and sometimes high-dimensional) state-action space, be tolerant to noise in demonstrations, and

they should capture risk characteristics of rewards rather than just their expectations. Furthermore, ideally we need IRL or IL methods that surpass the performance in demonstrations, rather than merely mimicking it and assuming that the demonstrated behavior is already nearly optimal. Such methods, including T-REX and D-REX, have been suggested only very recently, and we included examples of them in this chapter to motivate their adoption in finance.

Among applications of IRL to quantitative finance, we outlined applications that focus on inference of individual traders, and those that makes inference of the whole market dynamics.

For the first class of problems, as we demonstrated with the G-learner and GIRL algorithms in Sect. 10.3, IRL is able to recover the reward function of a RL agent or a human agent modelled as a RL agent. This suggests that IRL can be used for applications to robo-advising and inference of investor preferences.

For the second class of IRL problems that deal with a single market-wide agent, market dynamics can be related to the action of that agent. While multi-agent RL formulations of market dynamics are available and useful for certain applications, mapping onto one market-wide agent (the “Invisible Hand”) enables a view of IRL as a tool for constructing new market models.

12 Exercises

Exercise 11.1

- Derive Eq. (11.7).
- Verify that the optimization problem in Eq. (11.10) is convex.

Exercise 11.2

Consider the policy optimization problem with one-dimensional state- and action-spaces and the following parameterization of the one-step reward:

$$r(s, a) = -\log \left(1 + e^{-\theta \Psi(s, a)} \right),$$

where θ is a vector of K parameters, and $\Psi(s, a)$ is a vector of K basis functions. Verify that this is a concave function of a as long as basis functions $\Psi(s, a)$ are linear in a .

Exercise 11.3

Verify that variational maximization with respect to classifier $D(s, a)$ in Eq. (11.75) reproduces the Jensen–Shannon divergence (11.72).

Exercise 11.4

Using the definition (11.71) of the convex conjugate function ψ^* for a differentiable convex function $\psi(x)$ of a scalar variable x , show that (i) ψ^* is convex, and (ii) $\psi^{**} = \psi$.

Exercise 11.5

Show that the choice $f(x) = x \log x - (x + 1) \log \frac{x+1}{2}$ in the definition of the f-divergence (11.73) gives rise to the Jensen–Shannon divergence (11.72) of distributions P and Q .

Exercise 11.6

In the example of learning a straight line from Sect. 5.6, compute the KL divergence $D_{KL}(P_\theta || P_E)$, $D_{KL}(P_E || P_\theta)$, and the JS divergence $D_{JS}(P_\theta, P_E)$.

Exercise 11.7

- a. Show that minimization of the LS-GAN loss $V_{LSGAN}(D)$ in Eq. (11.87) produces the following relation for the optimal discriminator:

$$D(s, a) = \frac{a\rho_\pi + b\rho_E}{\rho_\pi + \rho_E}.$$

- b. Use this expression to show that minimization of the generator loss $V_{LSGAN}(G)$ in Eq. (11.87) is equivalent to minimization of the Pearson χ^2 divergence between the model density and a mixture of the expert and agent densities, as long as coefficients a, b, c satisfy the constraints $b - c = 1$ and $b - a = 2$.

Exercise 11.8

Compute the gradients of the AIRL objective (11.93) with respect to parameters θ and show that they coincide with the gradients obtained using adaptive importance sampling (11.39) to estimate gradients of the Max-Causal Entropy IRL objective function.

Appendix

Answers to Multiple Choice Questions

Question 1

Answer: 3.

Question 2

Answer: 1, 4.

Question 3

Answer: 1, 2.

Python Notebooks

This chapter is accompanied by a notebook comparing various IRL methods for the financial cliff walking problems. Further details of the notebook are included in the README .md file.

References

- Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. <https://arxiv.org/abs/1701.07875>.
- Baldovin, F., & Robledo, A. (2005). Parallels between the dynamics at the noise-perturbed onset of chaos in logistic maps and the dynamics of glass formation. *Phys. Rev. E*, 72. <https://arxiv.org/pdf/cond-mat/0504033.pdf>.
- Bertsimas, D., Gupta, V., & Paschalidis, I. (2012). Inverse optimization: A new perspective on the Black-Litterman model. *Operations Research*, 60(6), 1389–1403.
- Black, F., & Litterman, R. (1991). Asset allocation combining investor views with market equilibrium. *Journal of Fixed Income*, 1(2), 7–18.
- Boularias, A., Kober, J., & Peters, J. (2011). Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15* (pp. 182–189).
- Brown, D. S., Goo, W., Nagarajan, P., & Niekum, S. (2019). Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. arXiv:1904.06387.
- Brown, D. S., Goo, W., & Niekum, S. (2019). Better-than-demonstrator imitation learning via automatically-ranked-demonstrations. arXiv:1907.0397.
- Capponi, A., Olafsson, S., & Zariphopoulou, T. (2019). Personalized robo-advising: Enhancing investment through client interaction.
- Chasemaniour, S., Gu, S., & Zemel, R. (2019). Understanding the relation between maximum-entropy inverse reinforcement learning and behaviour cloning. ICLP.
- Dixit, A., & Pindyck, R. (1994). *Investment under uncertainty*. Princeton NJ: Princeton University Press.
- Dixon, M.F., & Halperin, I. (2020). G-Learner and GIRL: Goal Based Wealth Management with Reinforcement Learning, available at [https://papers.ssrn.com/sol3/papers.cfm?abstract\\$._\\$id=3543852](https://papers.ssrn.com/sol3/papers.cfm?abstract$._$id=3543852).
- Ewald, C. O., & Yang, Z. (2007). Geometric mean reversion: formulas for the equilibrium density and analytic moment matching. University of St. Andrews Economics Preprints.
- Finn, C., Christiano, P., Abbeel, P., & Levine, S. (2016). A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. arXiv:1611.03852.
- Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: deep inverse optimal control via policy optimization. arXiv:1603.00448.
- Fox, R., Pakman, A., & Tishby, N. (2015). Taming the noise in reinforcement learning via soft updates. In *32nd Conference on Uncertainty in Artificial Intelligence (UAI)*. <https://arxiv.org/pdf/1512.08562.pdf>.
- Fu, J., Luo, K., & Levine, S. (2015). Learning robust rewards with adversarial inverse reinforcement learning. arXiv:1710.11248. <https://arxiv.org/pdf/1512.08562.pdf>.
- Goodfellow, I., Pouget-Abadie, J., Mirza, B. X. M., Warde-Farley, D., Ozair, S., Courville, A., et al. (2014). Generative adversarial nets. *NIPS*, 2672–2680.
- Halperin, I., & Feldshteyn, I. (2018). Market self-learning of signals, impact and optimal trading: Invisible hand inference with free energy, (or, how we learned to stop worrying and love bounded rationality). https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3174498.

- Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems 29*. <http://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning.pdf>.
- Jaynes, E. (1957). Information theory and statistical mechanics. *Physical Review*, 106(4), 620–630.
- Kalakrishnan, M., Pastor, P., Righetti, L., & Schaal, S. (2013). Learning objective functions for manipulations. In *International Conference on Robotics and Automation (ICRA)*.
- Kampen, N. G. V. (1981). *Stochastic processes in physics and chemistry*. North-Holland.
- Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: a survey. *International Journal of Robotic Research*, 32(11), 1238–1278.
- Kramer, G. (1998). Directed information for channels with feedback. Ph.D. thesis, Technische Wissenschaften ETH Zürich.
- Lacotte, J., Ghavamzadeh, M., Chow, Y., & Pavone, M. (2018). Risk-sensitive generative adversarial imitation learning. <https://arxiv.org/pdf/1808.04468.pdf>.
- Landau, L., & Lifshitz, E. (1980). *Statistical physics. Course of theoretical physics. vol. 5 (3 ed.)*. Oxford: Pergamon Press.
- Levine, S., Popovic, Z., & Koltun, V. (2011). Nonlinear inverse reinforcement learning with Gaussian processes. *Advances in Neural Information Processing Systems*, 24.
- Liu, S., Araujo, M., Brunskill, E., Rossetti, R., Barros, J., & R. Krishnan (2013). Understanding Sequential Decisions via Inverse Reinforcement Learning. In *IEEE 14th International Conference on Mobile Data Management*.
- Mao, X., Li, Q., Xie, H., Lau, R., Wang, Z., & Smolley, S. P. (2016). Least squares generative adversarial networks. <https://arxiv.org/abs/1611.04076>.
- Marschinski, R., Rossi, P., Tavoni, M., & Cocco, F. (2007). Portfolio selection with probabilistic utility. *Annals of Operations Research*, 151(1), 223–239.
- Nguyen, X., Wainwright, M. J., & Jordan, M. I. (2010). Estimating divergence functionals and the likelihood ratio by convex risk minimization. *Information Theory. IEEE*, 56(11), 5847–5861.
- Nowozin, S., Scke, B., & Tomioka, R. (2016). F-GAN: training generative neural samplers using variational divergence minimization. <https://arxiv.org/abs/1606.00709>.
- Ortega, P., & Braun, D. A. (2013). Thermodynamics as a theory of decision-making with information processing costs. *Proceedings of the Royal Society A*. <https://doi.org/10.1098/rspa.2012.0683>. <https://arxiv.org/pdf/1204.6481.pdf>.
- Ortega, P. A., Braun, D. A., Dyer, J., Kim, K., & Tishby, N. (2015). Information-theoretic bounded rationality. <https://arxiv.org/pdf/1512.06789.pdf>.
- Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming*. New York, NY, USA: John Wiley & Sons, Inc.
- Ramachandran, D., & Amirv, E. (2007). Bayesian inverse reinforcement learning. *Proc. IJCAI*, 2586–2591.
- Reddy, S., Dragan, A. D., & Levine, S. (2019). SQL: imitation learning via regularized behavioral cloning. <https://arxiv.org/pdf/1905.11108.pdf>.
- Russell, S. (1998). Learning agents for uncertain environments. In *Proceeding of the Eleventh Annual Conference on Computational Learning Theory. COLT' 98*, ACM, New York, NY, USA, pp. 101–103.
- Shiarlis, K., Messias, J., & Whiteson, S. (2016). Inverse reinforcement learning from failure. In J. Thangarajan, Tuyls, K., Jonker, C., Marcella, S. (Eds.) *Proceedings of 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Singapore.
- Simon, H. (1956). Rational choice and the structure of the environment. *Psychological Review*, 63(2), 129–138.
- Sternberg, S. (2010). *Dynamic systems*. Dover Publications.
- Xu, L., Smith, J., Hu, Y., Cheng, Y., & Zhu, Y. (2015). A dynamic structural model for heterogeneous mobile data consumption and promotion design. Working paper, available at https://www.krannert.purdue.edu/academics/MIS/workshop/Xu-et-al_2015_DynamicMobileData.pdf.
- Yang, S. Y., Qiao, Q., Beling, P. A., Scherer, W. T., & Kirilenko, A. A. (2015). Gaussian process-based algorithmic trading strategy identification. *Quantitative Finance*, 15(10). <https://doi.org/10.1080/14697688..1011684>.

- Yang, S. Y., Yu, Y., & Almahdi, S. (2018). An investor sentiment reward-based trading system using Gaussian inverse reinforcement learning algorithm. *Expert Systems with Applications*, 114, 388–401.
- Yukalov, B. I., & Sornette, D. (2014). Self-organization in complex systems as decision making. *Advances in Complex Systems*, 3–4, 17.
- Ziebart, B., Bagnell, J., & Dey, A. K. (2013). The principle of maximum causal entropy for estimating interacting processes. *IEEE Transactions on Information Theory*, 59(4), 1966–1980.
- Ziebart, B., Maas, A., Bagnell, J., & Dey, A. (2008). Maximum entropy inverse reinforcement learning. *AAAI*, 1433–1438.

Chapter 12

Frontiers of Machine Learning and Finance



This final chapter takes us forward to emerging research topics in quantitative finance and machine learning. Among many interesting emerging topics, we focus here on two broad themes. The first one deals with unification of supervised learning and reinforcement learning as two tasks of perception-action cycles of agents. We outline some recent research ideas in the literature including, in particular, information theory-based versions of reinforcement learning, and discuss their relevance for financial applications. We explain why these ideas have interesting practical implications for RL financial models, where features are selected within the general task of optimization of a long-term objective, rather than outside of it, as is usually performed in “alpha-research.” The second topic presented in this chapter deals with using methods of reinforcement learning to construct models of market dynamics. We also introduce some advanced physics-based approaches for computations for such RL-inspired market models.

1 Introduction

Over the last decade, machine learning experienced a surge in popularity among both researchers and practitioners, and many interesting use cases for practical applications across different fields have emerged. Outside of finance, applications to digital services such as image recognition or speech recognition dominate the stream of research publications on supervised and unsupervised learning. Applications of reinforcement learning largely focus on robotics and video games, as evidenced by research efforts of such companies as Google’s DeepMind or OpenAI.

The ongoing research into machine learning continues to produce new methods that are intended to address the challenges of real-world applications better than their more “classical” predecessors. One example of such a new approach in machine learning are generative adversarial networks (GANs) that were discovered

by (Goodfellow et al. 2014) only in 2014, but has already been cited 12,000 times as of December 28, 2019. Many other successful algorithms of machine learning that were mentioned in this book such as variational autoencoders, networks with attention, deep reinforcement learning, etc. were developed over the last few years. Therefore, it may not be an exaggeration to claim that the pace of innovation in machine learning is at the scale of months rather than years. Clearly this means that any attempt to cover the most recent cutting edge research in a graduate-level textbook format would be futile, as such a book would be already outdated even before it completes its journey from writing to printing. Such a goal would be futile, but on the other hand it may not be one that we believe would be optimal for this book.

Precisely because our book is about ML in finance, rather than ML in general, among many exciting developments in machine learning across different applications, our choice of topics is driven by their potential applicability in finance. The idea is to give the reader both tips for further research and provide some nascent alternative directions that can be used for financial applications.

Respectively, in this chapter we will try to provide a brief overview of various new ideas that have been proposed in the recent literature but did not yet become known to many practitioners of machine learning in finance. In particular, in the first two sections we will talk about ideas from physics that might provide useful insights into financial problems amenable to machine learning methods.

The second theme of this final chapter is related to ideas of universality and unification. Machine learning is often presented as a fragmented set of algorithms that are all tuned to solve one particular task. For example, the search of predictive signals in quantitative trading amounts to finding observable (or computable) quantities that would be predictive of future asset returns or risk. This is typically formulated as a supervised learning problem that is solved by training on some historical data. As formulated, it is detached from the problem of optimal trading which is supposed to be based on the extracted signals. The latter problem of optimal trading given signals is usually addressed using methods of reinforcement learning. In this setting, trading signals are viewed as exogenous inputs computed outside of the RL model.

On the other hand, if we think of building a trading agent, the agent should be capable of both forecasting the future (via inferred trading signals), and discovering and executing an optimal policy. The first task is a task of *perception*, while the second task is an *action task*.

In the setting of a multi-step trading, the agent would perpetually switch between these two types of activities as it moves from one time step to another. In other words, it lives through many repetitions of a *perception-action cycle*.

While most current methods treat these two tasks (episodes of a perception-action cycle) separately, they are clearly subordinate: the task of producing an optimal trading policy is the main task, while the task of finding predictive signals is a secondary task. Ideally, we could think of machine learning approaches where these tasks would be integrated together, rather than tackled in isolation. In this chapter, we will present several recent approaches that aim at such a goal.

Chapter Objectives

This chapter will provide an overview of the frontiers of machine learning and reinforcement learning in finance covering:

- Market modeling beyond IRL using ideas from physics
 - New physics-based ideas in machine learning
 - Perception-action cycles
 - Unification of inference and planning
-

2 Market Dynamics, IRL, and Physics

Recall that Chap. 11 outlined how inverse reinforcement learning (IRL) can be used to model the dynamics of a market as a whole. To this end, we posed the problem of finding the optimal reward and policy of an “Invisible Hand” agent identified with the collective actions of all traders in the market. “Non-coherent” (or “zero-intelligence”) traders are considered a part of the environment for the “Invisible Hand” agent. As we discussed in Sect. 10.5, IRL for this setting with a quadratic reward (Markowitz utility) produces a linear optimal policy (11.154). Once the optimal policy is obtained as a linear function of the market capitalization of a stock (or just of the stock price), it can be plugged back into the dynamic price equation for the stock price. This produces a model of the market price of a stock with a non-linear (quadratic) drift term. This model provides a multivariate extension of the geometric mean reversion (GMR) model (Merton 1975; Dixit and Pindyck 1994; Ewald and Yang 2007) in the presence of signals.

The IRL-based model of (Halperin and Feldshteyn 2018) that we presented in Sect. 10.5 provides a mesoscopic description of market dynamics when viewed from the perspective of a market-wide agent. This is a view “from within” the market. In this approach, the agent’s actions u_t are adjustments of all positions (by all traders in the market) on a given stock at the beginning of the interval $[t, t + \Delta t]$. This produces a single-agent model of the market where the agent learns by a self-play, and the environment is produced by non-coherent “noisy traders” whose trades provide a stochastic component of the stock price evolution.

Instead of looking at market dynamics from “within” the market, we can alternatively consider these dynamics from *outside* of the market. Indeed, because we look at *all* traders in the market at once, it is natural to interpret their collective action u_t as an amount of *new capital* injected (or withdrawn, if it is negative) in the market by outside investors at the beginning of the interval $[t, t + \Delta t]$. As we will see shortly, such a “dual” view generalizes the IRL-based market dynamics model to describe not only a stable “growth” market phase (as implicitly assumed in Halperin and Feldshteyn (2018)), but also more realistic market regimes with corporate defaults and market crashes.

2.1 “Quantum Equilibrium–Disequilibrium” (QED) Model

Let X_t be a total capitalization of a firm at time t , rescaled to a dimensionless quantity of the order of one $X_t \sim 1$, e.g. by dividing by a mean capitalization over the observation period. We consider discrete-time dynamics described, in general form, by the following equations:

$$\begin{aligned} X_{t+\Delta t} &= (1 + r_t \Delta t)(X_t - cX_t \Delta t + u_t \Delta t), \\ r_t &= r_f + \mathbf{w}^T \mathbf{z}_t - \mu u_t + \frac{\sigma}{\sqrt{\Delta t}} \varepsilon_t, \end{aligned} \quad (12.1)$$

where Δt is a time step, r_f is a risk-free rate, c is a dividend rate (assumed constant here), \mathbf{z}_t is a vector of predictors with weights \mathbf{w} , μ is a market impact parameter with a linear impact specification, $u_t := u_t(X_t, \mathbf{z}_t)$ is a cash inflow/outflow from outside investors, and $\varepsilon_t \sim \mathcal{N}(0, 1)$ is white noise. Here the first equation defines the change of the total market cap¹ in the time step $[t, t + \Delta t]$ as a composition of two changes to its time- t value X_t . First, at the beginning of the interval, a dividend $cX_t \Delta t$ is paid to the investors, while they also may inject the amount $u_t \Delta t$ of capital in the stock. After that, the new capital value $X_t - cX_t \Delta t + u_t \Delta t$ grows at rate r_t . The latter is given by the second of Eqs. (12.1), where the term μu_t describes a linear trade impact effect. Note that u_t can be either zero or non-zero.

The reason that the same quantity u_t appears in both equations in (12.1) is simple. In the first equation, u_t enters as a capital injection $u_t \Delta t$, while in the second equation it enters via the market impact term μu_t because adding capital $u_t \Delta t$ means trading a quantity of the stock that is proportional to u_t . Using a linear impact approximation, this produces the impact term μu_t . As will be shown below, this term is critical even for very small values of μ because the limit $\mu \rightarrow 0$ of the resulting model is *non-analytic*.

In general, the amount of capital $u_t \Delta t$ injected by investors in the market at time t should depend on the current market capitalization X_t , plus possibly other factors (e.g., alpha signals). We consider a simplest possible functional form of u_t , without signals,

$$u_t = \phi X_t + \lambda X_t^2, \quad (12.2)$$

with two parameters ϕ and λ . Note the absence of a constant term in this expression, which ensures that no investor would invest in a stock with a strictly zero price. Also note that Eq. (12.2) can always be viewed as a leading-order Taylor expansion of a more general non-linear “capital supply” function $u(X_t, \mathbf{z}_t)$ that can depend on both X_t and signals \mathbf{z}_t . Respectively, parameters ϕ and λ could be slowly varying functions of signals \mathbf{z}_t . Here we consider a limiting case when they are treated as

¹Or, equivalently, the stock price, if the number of outstanding shares is kept constant.

fixed parameters, which may be a reasonable assumption when an economic regime does not change too much for an observational period in data.

Substituting Eq. (12.2) into Eqs. (12.1), neglecting terms $O(\Delta t)^2$, and taking the continuous-time limit $\Delta t \rightarrow dt$ we obtain the “Quantum Equilibrium–Disequilibrium” (QED) model (Halperin and Dixon 2020):

$$dX_t = \kappa X_t \left(\frac{\theta}{\kappa} - X_t - \frac{g}{\kappa} X_t^2 \right) dt + \sigma X_t (dW_t + \mathbf{w}^T \mathbf{z}_t), \quad (12.3)$$

where W_t is the standard Brownian motion, and parameters g , κ , and θ are defined as follows:

$$g = \mu\lambda, \quad \kappa = \mu\phi - \lambda, \quad \theta = r_f - c + \phi. \quad (12.4)$$

If we keep $\mu > 0$ fixed, the mean reversion parameter κ can be of either sign, depending on the values of ϕ and λ . If $\phi < \lambda/\mu$, then $\kappa < 0$, otherwise one for $\phi \geq \lambda/\mu$ we get $\kappa \geq 0$.

Equation (12.3) with $g = 0$ is known in physics and biology as the Verhulst population growth model with a multiplicative noise, where it is usually written in an equivalent form that can be obtained by a linear rescaling of the dependent variable X_t that makes the coefficient in front of term X_t^2 equal one.

Note that the higher-order terms in the drift in (12.3) are responsible for a possible saturation of the process. In population dynamics, this corresponds to a population competing for a bounded food resource. In a financial context, this spells a limited total wealth in a market without an injection of capital from the outside world.

2.2 The Langevin Equation

Equation (12.3) is a special case of the Langevin equation

$$dx_t = -U'(x_t)dt + \sigma x_t dW_t, \quad (12.5)$$

which describes an overdamped Brownian particle in an external potential $U(x)$ whose negative gradient gives a drift term in the equation, in the presence of a multiplicative noise.

The Langevin equation is named after Paul Langevin whose 1908 paper (Langevin 1908) extended the model of a free Brownian diffusion that was developed in physics by Albert Einstein in 1905. Einstein’s model is for a Brownian particle whose random dynamics is driven by interactions with other particles of the same sort. In Langevin’s extension of Einstein’s theory, the collective impact of *other* particles (e.g., an impact of large molecules on the dynamics of small molecules in a solution) or external fields (e.g., an electric field acting on a charged

particle) is codified into a potential $U(x)$. Such a potential is in general given by some non-linear function such as a polynomial.

A simple example is provided by a quadratic polynomial. Let $U(x) = \frac{m}{2}x^2$, where m is a parameter. Such a potential is called the *harmonic oscillator* potential in physics. Note that this potential is convex and has a unique minimum (stable point) at $x = 0$. Substituting it into the general Langevin equation (12.5), we obtain

$$dx_t = -mx_t dt + \sigma x_t dW_t. \quad (12.6)$$

This equation describes a particle (harmonic oscillator) with mass m that is subject to a combination of a deterministic linear term $\sim -mx_t$ and a proportional diffusion term $\sim \sigma x_t$.

In physics, the case with a harmonic oscillator potential $U(x) = \frac{m}{2}x^2$ is usually understood as a quadratic expansion of a more general non-linear potential

$$U(x) = u_0 + u_1 x + u_2 x^2 + \dots, \quad (12.7)$$

where all $O(x^3)$ terms and higher powers of the state variable x are neglected. It turns out that these higher-order terms are usually responsible for interactions defining the structure and stable states in complex physical systems. For this reason, in physics the case with a quadratic potential $U(x) \sim x^2$ is often referred to as a *non-interacting* case, though strictly speaking it has a harmonic oscillator (quadratic) interaction potential.

Respectively, the Langevin equation (12.5) is interpreted as an equation for the free harmonic oscillator subject to a multiplicative noise. The meaning of the word “free” refers here to the fact that interesting, non-trivial effects of interactions of Brownian particles with their media can be only captured by higher-order terms (a cubic, a quartic, etc.) in the potential $U(x)$.

2.3 The GBM Model as the Langevin Equation

We note that if we take the state variable x_t in the general Langevin equation (12.5) to be the stock price S_t , it looks very similar to the equation of the geometric Brownian motion (GBM) model

$$dS_t = \mu S_t dt + \sigma S_t dW_t. \quad (12.8)$$

In its turn, the GBM model can be considered a linear specification with $\mu(S_t) = \mu S_t$ and $\sigma(S_t) = \sigma S_t$ of a general Itô diffusion

$$dS_t = \mu(S_t) dt + \sigma(S_t) dW_t. \quad (12.9)$$

While both the Langevin equation (12.5) and the GBM equation (12.8) look very similar if we substitute $x_t = S_t$ and set $m = \mu$, there is a *critical* difference: the sign

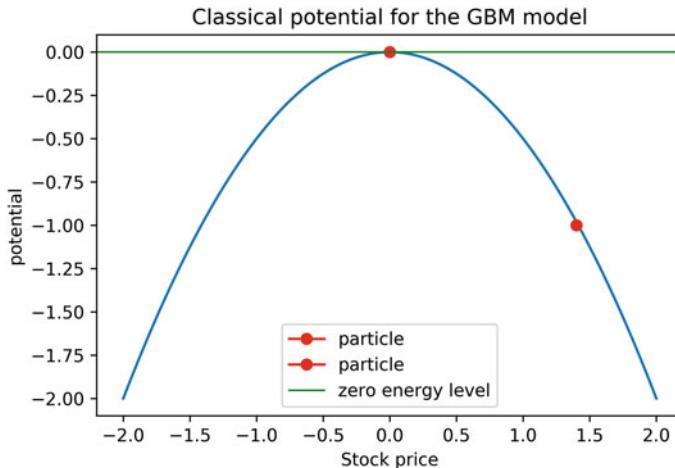


Fig. 12.1 The classical potential $U(x)$ corresponding to the geometric Brownian motion model. The firm stock price is described as the position of a particle influenced by this potential (the red dots). The potential corresponds to a harmonic oscillator with a negative mass, and describes an unstable system. A particle initially placed at position $x = 0$ will be unstable, and will quickly roll down the hill

of the drift term. If we interpret the linear drift μx_t of the GBM model as the negative gradient of a potential $U(x)$ as suggested by the Langevin equation, this corresponds to a harmonic oscillator with a *negative* mass. The potential for such a negative mass harmonic oscillator is an inverted parabola that does not have a point of stability at $x = 0$, as is the case for a harmonic oscillator with a positive mass $m > 0$. In other words, the GBM model describes dynamics that are *globally unstable*, as shown in Fig. 12.1. As a global instability can never be sustained indefinitely long, this indicates that the GBM model is *incomplete*, and a fuller model should have mechanisms to prevent such instability from proceeding indefinitely. As we will explain next, the QED model provides such a stabilization of dynamics.

2.4 The QED Model as the Langevin Equation

Unlike the GBM model that corresponds to globally unstable dynamics, the QED model in Eq. (12.3) corresponds to a quartic potential

$$U(x) = -\frac{1}{2}\theta x^2 + \frac{1}{3}\kappa x^3 + \frac{1}{4}gx^4. \quad (12.10)$$

If we compare this expression to the potential $U(x) = -\frac{1}{2}\mu x^2$ of the GBM model, the two expressions coincide if we set $\theta = \mu$ and take the limit $\kappa \rightarrow 0$, $g \rightarrow 0$.

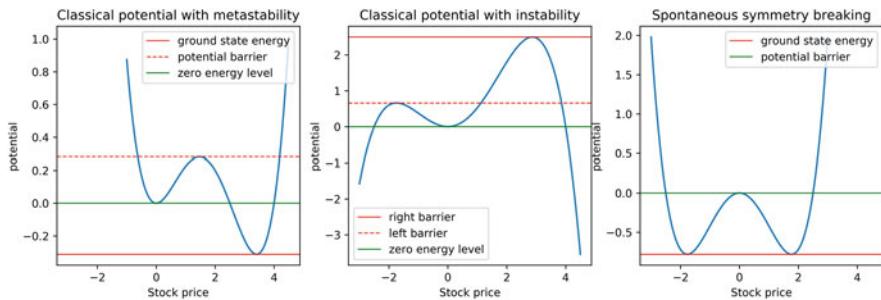


Fig. 12.2 Under different parameter choices in the QED model, the potential $U(x)$ takes different forms. A stable state of the system corresponds to a minimum of the potential. The potential on the left describes a metastable system with a local minimum at zero and a global minimum at $x = 3.3$. For the potential in the center, the state $x = 3.3$ becomes unstable, and the state $x = 0$ is metastable. The potential on the right has two symmetric minima, and the particle can choose any of them to minimize its energy. Such a scenario is called “spontaneous symmetry breaking” in physics

Therefore, the QED model can be considered an extension of the GBM model with a cubic drift.

Due to the presence of additional parameters κ and g controlling, respectively, the cubic and quartic non-linear terms in the potential $U(x)$, the latter can produce a wide variety of shapes, depending on the values of parameters, as illustrated in Fig. 12.2. As was shown in (Halperin and Dixon 2020), it is the potential in the left graph in Fig. 12.2 that leads to the most interesting dynamics of a stock market price. Instead of unstable dynamics of the GBM model, the QED model suggests that the dynamics can instead be *metastable*. Such dynamics are different from globally stable dynamics such as, e.g., the harmonic oscillator dynamics in that they *eventually* change, though the time for this change to occur may be long, or very long, depending on the parameters.

The explanation of how this happens is as follows. The potential shown on the left of Fig. 12.2 has a *potential barrier* between a metastable point at the bottom of the local well and the part of the potential for small values of x , where the motion against the gradient of the potential means a fall to the zero price level $x = 0$. Due to noise-induced fluctuations, a particle representing a stock with value x_t at time t placed initially to the right of the barrier can hop over to the left of the barrier. In physics, solutions of dynamics equations that describe such “barrier-hopping” transitions are called *instantons*. The reason for this nomenclature is that the transitions between the metastable state and the regime of instability (a “fall” to the zero level $x = 0$) happens almost instantaneously in time. What might take a long time though is the time for this hopping to occur: depending on model parameters, the waiting time can in principle even exceed the age of the observed universe.

In financial terms, such an event of hopping over the barrier en route to the zero level at $x = 0$ corresponds to a corporate bankruptcy (default). Due to the fact

that the GBM model corresponds to the inverted harmonic potential where the point $x = 0$ is unattainable, corporate defaults cannot be captured by the GBM model. In financial models that need to capture the presence of corporate defaults in the price dynamics, this is normally done by introducing additional degrees of freedom such as hazard rates.

In contrast, in the QED model corporate defaults are perfectly possible, and correspond to the instanton-type hopping transitions between different state of a metastable potential on the left of Fig. 12.2. As was shown in (Halperin and Dixon 2020), the QED model (12.3) admits a set of parameters that represent an equity model with a single degree of freedom (which is the stock price itself) that can simultaneously fit the stock price data and data on a credit default swap (CDS) referencing the same company (stock).

The QED model therefore amounts to non-linear Langevin dynamics with multiplicative noise that contains a white noise and colored noise components. The colored noise term describes signals used by investors. Such dynamics and related phase transitions are well studied in physics, see, e.g., Schmittmann and Zia (1995), den Broeck et al. (1997), Hinrichsen (2000). In particular, the problem of a noise-induced instanton transition from a metastable potential minimum is known in physics as the *Kramer's escape* problem, and the corresponding probability is given by a Kramer's escape rate formula. The Langevin equation and Kramer's escape rate relation were previously considered in the econophysics literature, in particular by Bouchaud and Cont (1998), Bouchaud and Potters (2004), and Sornette (2000, 2003) to describe market crashes.

2.5 Insights for Financial Modeling

The QED model of (Halperin and Dixon 2020) provides a number of interesting insights into financial modeling. Starting with the classical geometric Brownian motion (GBM) model, many other models of stock pricing such as local or stochastic volatility models typically have a linear drift term, while complexity of dynamics is ensured by making the noise (diffusion) term in the dynamics “more interesting,” i.e. non-linear, stochastic, etc.

On the contrary, the QED model points at the critical importance of a proper specification of the *drift term* to ensure the right behavior in the regime of very small and very large stock prices. In particular, unlike the GBM model or stochastic volatility models that are incompatible with the presence of corporate defaults in the market, the QED model is a parsimonious model with only one degree of freedom (the stock price itself) that enables defaults, and can be calibrated to *both* stock price data and CDS data. This enables the use of CDS data to better estimate long-term stock returns, and thus can be useful for long-term portfolio management. The innovation brought by the QED model in comparison to the GBM model and its direct descendants is that it incorporates capital inflows in the market and their impact on asset prices. Both these phenomena are well known to have a substantial

impact on the long-term behavior of asset prices, yet are not incorporated in most of traditional asset pricing models used in practice.

As shown by the QED model, incorporating both capital inflows and their price impact results in a quartic potential $U(x)$ or equivalently a cubic drift in the stock price. Such market frictions effects are often treated as “second-order” effects that could be handled, as long as the friction parameters κ, g remain small, as corrections to a regular behavior obtained in a friction-free limit $\kappa = g = 0$, using systematic expansions in small parameters known as perturbation theory methods. However, as was shown in (Halperin and Dixon 2020), the limit $\kappa, g \rightarrow 0$ is *discontinuous*: it does not exist as a limit of a smooth function. In particular, the instanton transitions (and hence corporate defaults) cease to exist in this limit. Instantons are essentially *non-perturbative* phenomena that cannot be uncovered using any finite order of perturbation theory in small friction parameters. Instead, instanton effects are treated in physics using *non-perturbative* methods that do not rely on perturbation theory in small parameters. See Halperin and Dixon (2020) for a review and the references to the relevant physics literature. The QED model suggests that non-perturbative phenomena are important in finance, and that methods developed in physics can be useful for modeling these phenomena.

2.6 Insights for Machine Learning

In addition to providing some new ideas into the modeling of dynamics of financial markets, the QED model may also offer useful insights for machine learning in general. Most importantly, it highlights the role of data that we normally do *not* have. Indeed, corporate defaults or market crashes are examples of *rare events*. When fitting a machine learning model to available stock price data, such rare events are usually underrepresented or altogether missing in the data. This creates *biased data*. For example, when traditional financial or machine learning models of equity returns are calibrated to available market data, stocks that have defaulted in the past are often removed from the dataset. The resulting dataset of equity returns becomes biased as it conveys no information about the mere existence of corporate defaults.

One conventional approach to compensate for missing or unavailable data in machine learning is to impose some generic regularization on a loss function. Common choices of regularization include, e.g., the L_2 and L_1 regularization. The QED model highlights the fact that the choice of the prior might be a fine art that is critical for enforcing the right behavior of the resulting model. To ensure the presence of a potential barrier separating the metastable and unstable states in the QED model, (Halperin and Dixon 2020) used what they referred to as the “Kramer’s regularization” as regularization that ensures that the potential has a barrier, so that the Kramer’s escape rate formula can be applied to compute the hopping probability. This suggests that more specialized regularization methods, and in particular methods preserving or maintaining certain static or dynamic *symmetries*, as opposed to the “generic” L_2 - and L_1 -regularization, can also be potentially interesting in other areas of machine learning.

3 Physics and Machine Learning

The previous section outlined an example where methods developed in physics are able to enrich a pure data-driven approach of the traditional machine learning. In this section, we continue with this theme in a slightly more general context, discussing the role of physics in machine learning as well as outline new ideas from physics that can be useful for problems amenable to machine learning methods.

Historically, many ideas among those that constitute the corpus of modern machine learning machinery have roots in physics. This list includes such fundamental concepts of machine learning as Monte Carlo methods, Boltzmann machines (that originate in the Ising model and other lattice models from physics), maximum entropy inference, energy-based models, etc. Most of these methods have been developed in physics in the nineteenth century and the first half of the twentieth century. However, physics has kept its own momentum over approximately the last 50 years, and some of these developments have emerged in the machine learning literature only very recently. In this section, we outline some of the most interesting work in this field.

3.1 *Hierarchical Representations in Deep Learning and Physics*

One of the central ideas of deep learning is hierarchical, multilayer compositions of non-linear functions. Deep learning processes the input data over multiple layers of non-linear transforms, providing a hierarchical representation of the original inputs. In deep convolutional networks, the data are hierarchically aggregated by combining inputs from a few neurons within a receptive field of a neuron in a higher layer, and proceeding in the same way to the next layer. This can be thought of as a hierarchical, multilayer coarse-graining of the initial data that produces gradually more abstract features when moving to higher layer in the network.

The procedure of coarse-graining the original raw features by proceeding hierarchically with multiple levels of abstractions has a clear parallel in physics under the name of a renormalization group. Coarse-graining of the renormalization group (RG) provides a systematic construction of the theory of large scales starting from an underlying microscopic theory. Therefore, RG can be interpreted as a mechanism to explain the emergence of large-scale structure, which is similar to deep learning.

RG methods were originally introduced in physics by Kadanoff and others for lattice models such as the Ising model and other models for discrete-valued systems of spins. The goal of real-space RG (Efrati et al. 2014) is to coarse-grain a given set of degrees of freedom \mathbf{X} in a position space in order to integrate out short-range fluctuations and retain only long-range correlations. This reduction results in a new “effective” theory with a Hamiltonian function defined on the space of coarse-grained variables. Note that the functional form of the Hamiltonian is preserved at

the coarse-graining transformation: if $H(\mathbf{X}, \theta)$ is the Hamiltonian in the original variables \mathbf{X} with parameters θ , then the new coarse-grained Hamiltonian $H(\mathbf{X}', \theta')$ will have the same functional form, but different parameters (or coupling constants of the Hamiltonian, using physics terms). An iterative application of this procedure gives rise to recursive relations between coupling constants of the Hamiltonian at successive RG steps. These relations are referred to as the *RG flow equations*—formalizing the relationship between effective theories at different length scales.

We can see that the real-space RG methods of physics appear similar to the input processing in deep neural network, as both approaches proceed in a hierarchical way. There are also some differences: with RG approaches in physics, connections between subsequent representations should be such that the more abstract representation preserves the partition function. Therefore, RG flow equations determine the structure of a network, rather than the dynamics of learning which is the focus in deep learning applications.

Similarities and differences between RG and deep learning are interesting to explore because, to date, theoretical explanations of deep learning are still in their infancy. One may therefore only speculate that deep learning performs a sophisticated coarse-graining. A number of recent works in the physics community pursued the possibility that RG may provide a useful framework for a theoretical analysis of deep learning. For example, de Mello Koch et al. (2019) used the Ising model, a statistical mechanics model for a magnet, to train an unsupervised restricted Boltzmann machine (RBM). They compared patterns generated by the trained RBM to the configurations generated through a RG treatment of the Ising model, and found some similarities between the RG flow and RBM flow. In particular, they looked at correlation functions between hidden and visible neurons as they turn out to be capable of diagnosing RG-like coarse-graining. Numerical experiments in (de Mello Koch et al. 2019) found the presence of RG-like patterns in correlation functions computed using the trained RBMs. This supports the idea that pursuing similarities between physics and deep learning can provide new insights into the theoretical explanation of deep learning.

3.2 Tensor Networks

One of the remarkable success stories of statistical physics in the last 30 years was the development of methods that use tensor decompositions for the analysis of quantum spin systems described by lattice models.

To explain the essence of this development, let us first revisit the definition of tensors themselves. Tensors are essentially multi-dimensional arrays that extend the notion of a matrix as a two-dimensional (2D) array or 2D table to multiple dimensions. Tensors can be considered “multi-view tables.” An N -dimensional tensor \mathbf{X} is represented by its elements X_{i_1, \dots, i_N} which is indexed by a N -dimensional integer-valued index $\mathbf{i} = (i_1, \dots, i_N)$. A regular matrix can be viewed

as a tensor of dimension 2, a vector is a tensor of dimension 1, and a scalar is a tensor of dimension zero.

Most of the data available for machine learning is naturally represented by tensors, and it is not by accident that Google’s library for machine learning is called TensorFlow—it operates with tensors as the most generic inputs. In finance, tensors provide the most natural format for data. For example, historical stock data for a given universe of stocks can be represented by a 3D tensor where the first index is for the time step, the second index is for the stock, and the third index enumerates stock features. If the size of the third dimension is one, so that it can only carry one number, we can place the current stock return in this element, which will produce the standard data matrix of returns. If the size of the third dimension of the tensor is larger than one, it can keep other features of the stock/company such as fundamentals ratios, sentiment scores, etc.

While input data may naturally exhibit a tensor format, it remains an open question as to how we proceed with processing such data. Most of the modern machine learning methods, including deep learning approaches, rely internally on a vector, rather than tensor, representation of data. For example, a feedforward neural network trained on 2D images (i.e., 2D tensors), such as the MNIST handwritten numbers, first converts them into vectors (1D tensors) by stacking columns of input 2D matrices. However, such a procedure may break the correlation structure of components of inputs in the original data. For example, in the MNIST example, pixels in neighboring cells in the 2D image can strongly correlate but this correlation can be lost (or, better to say, hidden) when the 2D matrix is converted to a vector by stacking its columns.

Fortunately, methods developed in applied mathematics provide alternative, and more tensor-focused methods of analysis of tensor-valued data. They are known as *tensor decompositions*. Tensor decompositions can be considered as multi-dimensional generalizations of matrix decomposition methods such as the singular value decomposition (SVD). While SVD decomposes a matrix (2D tensor) into a sum of direct products of eigenvectors (1D tensors), tensor decomposition provides a systematic framework for similar decompositions of N -dimensional tensors into lower-dimensional tensors. Classical methods of tensor decomposition such as the CP and Tucker decompositions (see, e.g., (Kolda and Bader 2009) for a review) can be considered multi-dimensional generalizations of the SVD and PCA methods of linear algebra (i.e., 2D-tensors methods).

In physics, interest in tensor decompositions was inspired by the need for numerical implementations of RG methods. A system of N quantum spins can be represented by a tensor of order N . Such a tensor can be obtained, e.g., by taking a direct product of individual wavefunctions $\Psi_k(x)$ with $k = 1, \dots, N$. For numerical implementations, the cost of storing a full N -dimensional tensor of data obtained by a direct product of all individual states would be prohibitive once N exceeds tens or hundreds, and therefore some ways to compress such data would be required.

Tensor decompositions in physics generally seek a low-dimensional data representation of data, where dimension reduction is achieved in a similar way to the

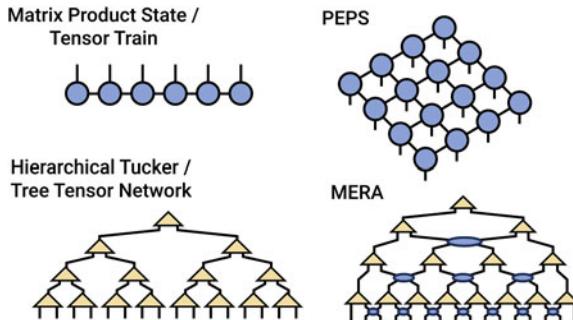


Fig. 12.3 Different architectures for tensor networks: (top left) A Matrix Product State (MPS) or tensor-train (TT) network factorizes a tensor into a chain product of three-index tensors; (top right) A PEPS (projected entangled pair states) tensor network generalizes the one-dimensional MPS/TT network to a network on an arbitrary graph; (bottom left) A tree tensor (hierarchical Tucker) network stacks tensors in the Tucker format; (bottom right) A MERA (multi-scale entanglement renormalization ansatz) tensor network is a tree network augmented with unitary disentangler operations between branches at each scale. Source: <http://tensornetwork.org>, with permission for use granted by Miles Stoudenmire

SVD and PCA methods, by expansion into a series of lower-order components. A tensor network is a factorization of an order N tensor into the contracted product of low-order tensors. Tensor networks break the curse of dimensionality by allowing operations such as contracting very high-order tensors or retrieving their components to be accomplished with polynomial cost by manipulating the low-order factor tensors.

In particular, Matrix Product State (MPS) decompositions known in applied mathematics as tensor-train decompositions have found many useful applications in physics as tools to numerically implement RG. The MPS (or tensor-train) decomposition represents a high-dimensional tensor as a sum of products of low-rank tensors. While the MPS decomposition corresponds to a tensor network with a linear architecture, there are also hierarchical versions of tensor networks as in Fig. 12.3.

Tensor networks provide many interesting insights into deep learning. Similar to deep learning, tensor networks build a hierarchical representation of data by constructing progressively more abstract features by coarse-graining features from the previous abstraction level. The most interesting difference is that while, in general, the information in a neural network is stored in a full-rank tensor of network weights at different layers, in a tensor network this information is compressed by storing all relevant data in low-order rank tensors. Similar to the PCA, such a decomposition does not provide an exact recovery of the input data, but rather stores its “de-noised” version. De-noising the data thus becomes a part of data compression within a tensor network.

Among many interesting recent proposals on applications of tensor networks to data analysis and machine learning, we would like to outline one particular idea that offers a way to use tensor networks to construct features in an unsupervised way (Stoudenmire 2017).

The main idea of (Stoudenmire 2017) is to use tensor networks as a way to produce high-level features starting with local feature maps. It goes as follows. We assume that input data are described by some raw d -dimensional features \mathbf{x} , and we seek feature maps $\Phi(\mathbf{x})$ that map N raw inputs \mathbf{x} of onto a space of dimension d^N with a tensor product structure. Such a map can be constructed starting with *local feature maps* $\phi^{s_j}(\mathbf{x}_j)$, where $s_j = 1, \dots, d$. The full feature map is then constructed by taking the direct product of local feature maps:

$$\Phi^{s_1 s_2 \dots s_N}(\mathbf{x}) = \phi^{s_1}(\mathbf{x}_1) \phi^{s_2}(\mathbf{x}_2) \cdots \phi^{s_N}(\mathbf{x}_N). \quad (12.11)$$

Given the feature map (12.11), a data model can be represented as an expansion

$$f(\mathbf{x}) = \sum_{s_1, s_2, \dots, s_N} W_{s_1 s_2 \dots s_N} \phi^{s_1}(\mathbf{x}_1) \phi^{s_2}(\mathbf{x}_2) \cdots \phi^{s_N}(\mathbf{x}_N). \quad (12.12)$$

Here \mathbf{W} is a tensor of order N that stores d^N coefficients of the expansion. The expression (12.12) is a contraction of two order N tensors. Clearly, manipulating or even storing d^N parameters quickly becomes impractical as N increases. A solution proposed in (Stoudenmire 2017) was to approximate the optimal weights \mathbf{W} by a tensor network. A simultaneous tensor network coarse-graining of local feature maps and the coefficient tensor \mathbf{W} was shown to produce a layered, hierarchical way of producing high-level features in an unsupervised way.

As discussed in (Stoudenmire 2017), this procedure resembles the hierarchical feature construction in deep neural networks. The difference is that with tensor networks, tensor contraction operations are all linear operations, the only source of non-linearity is in the construction of local feature maps $\phi^{s_j}(\mathbf{x}_j)$ ((Stoudenmire 2017) uses polynomial features). As linear tensor contraction operations are fully theoretically controllable, this provides a controllable approach to extracting abstract features from data. This is different from neural networks that are obtained by stacking multiple non-linear layers that are harder to control theoretically.

Tensor networks thus offer a principled way to construct abstract high-level features starting from arbitrary local feature maps. This may be useful for both supervised learning and reinforcement learning. In particular, as we discussed on a few occasions in Chaps. 9 and 10, a good choice of basis functions is important in applications of reinforcement learning to multi-dimensional control tasks. Tensor networks suggest a way to construct such bases in a bottom-up fashion starting with local feature maps.

3.3 *Bounded-Rational Agents in a Non-equilibrium Environment*

Another area of physics where interesting recent developments have potential applications in machine learning are non-equilibrium processes in mesoscopic and macroscopic systems studied in statistical physics. We recall that classical machine learning methods such as energy-based models, Boltzmann machines, maximum entropy method, etc. are all based on concepts of equilibrium statistical mechanics developed by physicists starting from the work of Ludwig Boltzmann in the nineteenth century.

In thermodynamics and statistical mechanics, non-equilibrium processes are often modeled by specifying a time-dependent external parameter $\lambda(t) \in [0, 1]$ that determines how the energy function $E_\lambda(x)$ changes over time. For example, with a linear switching on two potentials with energies $E_0(x)$ and $E_1(x)$, the energy would be $E_\lambda(x) = E_0(x) + \lambda(E_1(x) - E_0(x))$. When the change in the parameter λ is done infinitely slowly (i.e., quasi-statically), the system probability distribution follows the path of equilibrium distributions $p_\lambda(x) = \frac{1}{Z_\lambda} e^{-\beta E_\lambda(x)}$ for any value of λ . However, when the switching of the parameter λ is done in finite time, the non-equilibrium path of probability distributions can be in general different from the equilibrium path.

To think of implications of the equilibrium assumption in finance, consider the problem of building a financial trading agent. Let \mathbf{z}_t be market signals used by the agent for trading decision-making. The implicit assumption made in many models is that upon changes of trading signals \mathbf{z}_t , the market has enough time to “equilibrate” and find a new stationary or quasi-stationary state where market prices fully absorb the new information in the signals \mathbf{z}_t .

But this assumption should not necessarily hold in all market scenarios. When there is a fast change in the environment due to changes in signals \mathbf{z}_t that play the role of the external parameter $\lambda(t)$ in thermodynamics, market dynamics are unable to follow the path of equilibrium distributions, and instead evolve in a non-equilibrium fashion. The difference between the equilibrium and non-equilibrium can be clarified using the concept of characteristic times. If the characteristic time τ_z of relaxation under changes of signals \mathbf{z}_t is larger than the frequency of trading, the assumption of the equilibrium does not apply anymore—the dynamics will be non-equilibrium in such a scenario. It is therefore of interest to financial applications to consider decision-making agents in a changing, non-equilibrium environment.

As we already discussed in Sect. 10.5 in Chap. 11, the concept of bounded-rational agents, initially proposed by Simon (1956) is a useful paradigm for finance. It replaces perfectly rational agents of the von Neumann–Morgenstern expected utility approach that assumes that market agents are fully rational, and each of them maximizes a well-defined utility (reward) function, by the concept of an agent with constrained information-processing resources. Due to a lack of computational resources, a bounded-rational agent is unable to find a perfect action according to a given utility function.

The bounded rationality theory of Simon has been enriched with tools from information theory and thermodynamics (Tishby and Polani 2011; Ortega and Braun 2013; Ortega et al. 2015). With this approach, a bounded-rational agent maximizes an augmented reward function that incorporates the information cost of updating from some prior (reference) policy π_0 . When the relative weight of two components in the augmented reward function is determined by the inverse temperature parameter β , this parameter also controls the degree of rationality of the agent, such that in the high-temperature limit $\beta \rightarrow 0$, the most optimal behavior for the agent is not to update the “prior” policy π_0 at all, i.e. to behave in a fully irrational manner. For non-zero values of β , the optimal solution to the problem of optimization of a tradeoff between maximization of the reward and minimization of information costs takes the form of a Boltzmann distribution analogous to equilibrium distributions in statistical physics. This implies that the decision-making process of the agent can be understood as a change from a prior policy π_0 to a posterior policy π , where the change of the policy is triggered by a change of the environment.

Mathematically, as we saw in Sect. 10.5, augmenting the reward function with the KL information cost is equivalent to regularization of the value and action-value function by the KL entropy as done in G-learning. In other words, the approach of G-learning corresponds to the assumption of equilibrium dynamics for a bounded-rational agent. With this approach, the optimal action of the agent is determined by a Boltzmann-like policy, and is driven by the difference of free energies upon a change of the environment. It is therefore of interest to consider an extension of this approach to the case of a bounded-rational agent that operates in a changing, non-equilibrium environment.

This problem was addressed in (Grau-Moya et al. 2018). When the environment is out of equilibrium, a bounded-rational agent is not able to fully utilize the difference in free energies of equilibrium states, and some of its utility gets lost by dissipation and heating of the environment. Using recent results known in non-equilibrium thermodynamics as generalized fluctuation theorems and Jarzynski equalities that extend the thermodynamic work relations to non-equilibrium systems, (Grau-Moya et al. 2018) obtained relations for free energy changes of a bounded-rational agent upon non-equilibrium changes of the environment, and related them to the amount of dissipated energy. While the analysis in (Grau-Moya et al. 2018) was in a one-step utility optimization setting, extending this work to a multi-period case might be important for RL because the free energy directly relates to the value function—which is the quantity that is maximized in RL.

4 A “Grand Unification” of Machine Learning?

In this book we considered three main types of machine learning: supervised learning, unsupervised learning, and reinforcement learning (with IRL as a sub-class of reinforcement learning). The first two types correspond to the class of *perception*

tasks for an artificial agent. Reinforcement learning corresponds to a different class of tasks for an artificial agent that are called *action tasks*.

So far, we largely presented these three types of machine learning as separate tasks that can at times utilize each other's methods. For example, reinforcement learning can use features obtained using supervised or unsupervised learning. There are also benefits that reinforcement learning brings to supervised learning. In particular, reinforcement learning methods can perform real-time cost-efficient feature selection for supervised learning. More specifically for financial applications, recurrent themes of this book are a unification of methods of econometrics and machine learning for supervised learning in finance, and a unification of methods of stochastic optimal control and reinforcement learning for portfolio optimization tasks. All these would be examples of a mutual penetration of methods across different branches of machine learning, and unification with tools developed outside of machine learning.

In this section, we will provide a different view of an interplay between different types of machine learning, and discuss why it is beneficial to consider these tasks of machine learning *jointly* rather than separately. By this, we mean not just a technical application of each other's methods for achieving a narrowly formulated goal, but rather their *unified view* within a higher-level of abstraction for modeling of artificial agents.

To illustrate such potential benefits, consider the problem of building an artificial agent for trading in equity markets. For such an agent, the supervised learning part of training amounts to finding some “signals” (functions of observable market data) that would be both predictive of their own future values (i.e., have a sufficiently high autocorrelation), and predictive of future asset returns. The supervised learning problem is then solved by training on some historical data.

The main problem with this approach is that it is not directly tied to the ultimate goal of the agent, which is to make a profit by trading. A trading signal obtained using supervised learning can be both predictive of its own future value, and correlate with equity returns, and yet not be very practical to use. For example, a strategy relying on such a signal may produce too high transaction costs that would subtract from profits expected according to the trading signal analysis. But handling such potential problems is outside of the supervised learning algorithms, because transaction costs arise only from trading, i.e. actions, which are *not* considered a part of the problem.

This means that as formulated, the supervised learning part of the trading agent's learning is detached from the problem of optimal trading which is supposed to be based on the extracted signals. Once obtained using supervised learning, trading signals are then used as exogenous inputs for a reinforcement learning agent who attempts to optimize a trading strategy. Therefore, with this approach, the perception task of forecasting the future via inference of trading signals and the action task of trading optimally are viewed in isolation.

On the other hand, in the context of a multi-period trading, the agent alternates between these two types of activities, as it moves from one time step to another. In other words, it lives through many repetitions of a *perception-action cycle*.

While most current methods treat the perception and action tasks as separate elements of a perception-action cycle, they have a clear hierarchical structure: the task of producing an optimal trading policy is the main task, while the task of finding predictive signals is a secondary task. This implies that the trading agent is free to design a state representation and laws of dynamics that are specifically tuned to agent’s ultimate goals, rather than build models of dynamics that might be “right” abstractly but not be helpful in achieving the goal. In this section, we will outline several recent approaches that aim at providing an integrated, “unified” view of sub-tasks of perception and action for artificial agents, instead of treating them separately.

4.1 Perception-Action Cycles

In the research literature on intelligent behavior in organisms, the perception-action cycle describes the circular flow of information between an organism and its environment in the course of a sensory guided sequence of actions towards a goal. The same concept can also be applied to describe interactions of an artificial agent with its environment. We can therefore approach this problem in general terms, referring to both biological organisms and artificial agents simply as “agents.”

With feedback to the environment from actions of the agent, the cycle introduces complex dependencies between perception and action tasks. As actions change the environment, perception is *not* passive, but rather depends on actions that were selected earlier by the agent. For a living organism, this implies that it can control, to some extent, which sensor inputs it will experience in the future, or decide which sensor inputs can be deemed irrelevant for planning. For an artificial agent such as a trading agent, the role of sensory inputs is played by trading signals \mathbf{z}_t . Therefore, within a cycle-focused view of perception and action tasks, they become tightly intertwined.

As was shown in (Tishby and Polani 2011; Ortega and Braun 2013; Ortega et al. 2015), information-theoretic methods provide a unified and model-independent way to describe such an interplay between perception and action within a perception-action cycle. Within this approach, the information flow of the cycle is viewed as a *bi-directional* information passing process.

First, there is an information flow from the environment to the agent. In our example with a trading agent, this would be market information that is used to construct trading signals \mathbf{z}_t .

Second, there is information that is passed from the agent to the environment. Again, in the financial context it is easy to find an example of such information transfer. When an agent takes a large position, other market participants often conceive it as an evidence that the first agent possessed superior information that allegedly facilitated the trade. Therefore, they might correct their estimates and trading decisions accordingly—which jointly corresponds to a change of the market environment.

4.2 Information Theory Meets Reinforcement Learning

As we discussed on several occasions in both the previous chapters and this chapter, information-theoretic tools are very useful for problems of reinforcement learning. Recall that one example that we considered in Chap. 10 is G-learning. It provides a probabilistic extension of Q-learning that incorporates information-processing constraints for agents into their policy optimization (planning) objectives.

While having solid roots in information theory, G-learning offers valuable practical benefits in being a generative model that is capable of processing noisy high-dimensional data. As we saw in Sect. 10.5 and in the previous section, adding KL penalties to augment the reward as performed in G-learning can also be viewed as a way to model a bounded-rational agent, where the amount of rationality is controlled by the magnitude of information-processing costs. This provides a practical and principled information-theoretic implementation of the concept of a bounded rational agent of Simon (1956).

G-learning is based on incorporating information-processing costs constraints into decision-making of the agent. From the point of view of bi-directional information flow between the agent and the environment, G-learning tackles the information flow from the environment to the agent. It is therefore of interest to extend this framework in order to also include the second information flow, from the agent to the environment.

Such an extension of the information-based reinforcement learning was recently developed in (Tiomkin and Tishby 2018). The authors considered the feedback interaction between the agent and the environment as consisting of two asymmetric information channels. Directed (causal) information from the environment constrains the maximum expected reward that is considered in the standard RL setting. Tiomkin and Tishby developed a Bellman-like recursion equation for the causal information between the environment and the agent, in the setting of an infinite-horizon Markov Decision Process problem. This relation can be combined with the Bellman recursion for the value function into a unified Bellman equation that drives both causal information flow and value function of the agent. As was pointed out by (Tiomkin and Tishby 2018), this approach has potentially important practical applications for the design criteria of intelligent agents. More specifically, an information-processing rate of a “brain” of the agent (i.e., of its processor) should be higher than the minimum information rate required to solve the related MDP problem (Tiomkin and Tishby 2018).

The approach of (Tiomkin and Tishby 2018) applies to living organisms or artificial agent operating with an infinite-horizon setting. For financial applications, an infinite-horizon setting can never be exact, but might be a good approximation for problems involving many time steps. For other tasks that involve a multi-step decision-making with a fixed and small number of time steps, this may not be the most suitable setting.

4.3 Reinforcement Learning Meets Supervised Learning: *Predictron, MuZero, and Other New Ideas*

A related but different approach to building agents which are able to plan for long-term goals that incorporate decision-making extended over many steps is being pursued by researchers at Google’s DeepMind. Unlike the work of (Tiomkin and Tishby 2018) that considers an infinite-horizon decision-making, this research focuses on finite-horizon problems of multi-step planning based on a look-ahead search. Examples of such planning problems are provided by video games such as Atari 2600 games, or traditional games such as chess or Go.

The common feature of such planning problems is that they all require a search for outcomes of an agent’s actions many steps into the future. In the setting of reinforcement learning, this describes learning with delayed rewards, where for all intermediate time steps, immediate rewards are all zeros, and it is only the reward obtained at the very last step (e.g., a checkmate in the chess game) that defines the overall score of the game. Planning for such problems should include a scenario analysis of agent’s actions over multiple steps into the future.

Methods pursued by researchers at DeepMind are focused on solving such finite-horizon planning tasks, and typically operate within neural network-based deep reinforcement learning architectures. With these approaches, sufficiently complex (deep) neural networks are used as universal function approximators for the value function and/or policy function for a reinforcement learning agent. For types of experiments that are commonly used to explore the performance of these methods in the broader machine learning community, DeepMind and other researchers often use simulated video games environments, or simulated games such as chess or Go, or alternatively simulated environments for physical robots such as MuJoCo.

The approach taken by DeepMind belongs in the class of model-based reinforcement learning where the task is to develop an end-to-end predictor of the value function. The main idea of this approach is to construct an abstract MDP model that is constrained by the requirement that planning in the abstract MDP is equivalent to planning in the real environment. This is called *value equivalence*, and amounts to the requirement that the cumulative reward through the trajectory in the abstract MDP should match the cumulative reward of a trajectory in the real environment.

This idea was first implemented in the *predictron* that constructs neural network-based value equivalent models for predicting the value function (Silver 2017). The innovation of this approach is that beyond ensuring value equivalence, it does not constrain in any way the resulting abstract MDP. In particular, states of such MDP are considered hidden states that should not necessarily provide a condensed view of the observed state, as is often assumed in unsupervised learning approaches. Likewise, there is no requirement that transitions between states in the abstract MDP should match transitions in the real environment. The only purpose of the abstract MDP is to help finding the optimal solution of the planning problem.

This approach was further extended in DeepMind’s MuZero algorithm that combines learning a model of the world with Monte Carlo tree search to achieve

super-human performance for both Atari 2600 video games and the games of Go, chess, and shogi without any knowledge of the game rules, and without any changes in the architecture of the agent (Schrittwieser 2017). The MuZero agent learns through self-play, without any supervision from a teacher. Learning is end-to-end, and involves simultaneous learning of the hidden state, transition probabilities for the hidden state, and the planning optimization algorithm.

The predictron and MuZero agents thus implement a unified approach to supervised learning and reinforcement learning where the task of supervised learning is subordinate to the ultimate goal of planning optimization. Exploring similar approaches for problems of financial planning and decision-making is a very promising future direction for machine learning in finance.

References

- Bouchaud, J., & Cont, R. (1998). A Langevin approach to stock market. *Eur. Phys. J. B*, 6(4), 543–550.
- Bouchaud, J., & Potters, M. (2004). *Theory of financial risk and derivative pricing*, 2nd edn. Cambridge: Cambridge University Press.
- de Mello Koch, E., de Mello Koch, R., & Cheng, L. (2019). Is deep learning an RG flow? <https://arxiv.org/abs/1906.05212>.
- den Broeck, C. V., Parrondo, J., Toral, R., & Kawai, R. (1997). Nonequilibrium phase transitions induced by multiplicative noise. *Physical Review E*, 55(4), 4084–4094.
- Dixit, A., & Pindyck, R. (1994). *Investment under uncertainty*. Princeton NJ: Princeton University Press.
- Efrati, E., Wang, Z., Kolan, A., & Kadanoff, L. (2014). Real-space renormalization in statistical mechanics. *Review of Modern Physics*, 86, 647–667.
- Ewald, C. O., & Yang, Z. (2007). *Geometric mean reversion: formulas for the equilibrium density and analytic moment matching*. University of St. Andrews Economics Preprints.
- Goodfellow, I., Pouget-Abadie, J., Mirza, B. X. M., Warde-Farley, D., Ozair, S., Corville, A., et al. (2014). Generative adversarial nets. *NIPS*, 2672–2680.
- Grau-Moya, J., Kruger, M., & Braun, D. (2018). Non-equilibrium relations for bounded rational decision-making in changing environments. *Entropy*, 20, 1. <https://doi.org/10.3390/e2001001>.
- Halperin, I., & Dixon, M. (2020). “Quantum Equilibrium-Disequilibrium”: Asset price dynamics, symmetry breaking, and defaults as dissipative instantons. *Physica A: Statistical Mechanics and Its Applications*, 537. <https://doi.org/10.1016/j.physa.2019.122187>.
- Halperin, I., & Feldshteyn, I. (2018). Market self-learning of signals, impact and optimal trading: invisible hand inference with free energy, (or, how we learned to stop worrying and love bounded rationality). https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3174498.
- Hinrichsen, H. (2000). Nonequilibrium critical phenomena and phase transitions into absorbing states. *Advances in Physics*, 49(7).
- Kolda, T., & Bader, B. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3), 455–500.
- Langevin, P. (1908). Sur la théorie du mouvement brownien. *Comptes Rendus Acad. Sci. (Paris)*, 146, 530–533.
- Merton, R. C. (1975). An asymptotic theory of growth under uncertainty. *Review of Economic Studies*, 42(3), 375–393.
- Ortega, P., & Braun, D. A. (2013). Thermodynamics as a theory of decision-making with information processing costs. *Proceedings of the Royal Society A*. <https://doi.org/10.1098/rspa.2012.0683>. <https://arxiv.org/pdf/1204.6481.pdf>.

- Ortega, P. A., Braun, D. A., Dyer, J., Kim, K., & Tishby, N. (2015). Information-theoretic bounded rationality. <https://arxiv.org/pdf/1512.06789.pdf>.
- Schmittmann, B., & Zia, R. (1995). *Statistical mechanics of driven diffusive systems*: Vol 17: *Phase transitions and critical phenomena*. In C. Domb, & J.L. Lebowitz (Ed.). Academic Press.
- Schrittwieser, J. (2017). Mastering atari, go, chess and shogi by planning with a learned model. <https://arxiv.org/abs/1911.08265>.
- Silver, D. (2017). The predictron: end-to-end learning and planning. In *ICML'17 Proceedings of the 34th International Conference on Machine Learning* (Vol. 70, pp. 3191–3199).
- Simon, H. (1956). Rational choice and the structure of the environment. *Psychological Review*, 63(2), 129–138.
- Sornette, D. (2000). Stock market speculations: spontaneous symmetry breaking of economic valuation. *Physica A*, 284(1–4), 355–375.
- Sornette, D. (2003). *Why stock markets crash*. Princeton: Princeton University Press.
- Stoudenmire, E. M. (2017). Learning relevant features of data with multi-scale tensor networks. *Quantum Science and Technology*, 3(3). <https://iopscience.iop.org/article/10.1088/2058-9565/aaba1a/meta>, available at <https://arxiv.org/pdf/1801.00315.pdf>.
- Tiomkin, S., & Tishby, N. (2018). A unified Bellman equation for causal information and value in Markov decision processes. <https://arxiv.org/abs/1703.01585>.
- Tishby, N., & Polani, D. (2011). *Information theory of decisions and actions* (pp. 601–636). Perception-Action Cycle. New York, NY, USA: Springer.

Index

A

Absolute error, 54
Action-value function, 294
Actor-critic, 310
ADAM, 158
Adaptive filtering, 234
Adversarial inverse reinforcement learning (AIRL), 472–474, 476
Adversarial IRL, 459
AI, *see* Artificial intelligence (AI)
AIC, *see* Akaike's information criteria (AIC)
AIRL, *see* Adversarial inverse reinforcement learning (AIRL)
Akaike's information criteria (AIC), 64
Alpha-RNN, 249
Alternative data, 4
Area Under the Curve (AUC), 213
ARMA model, 202
AR process, 194
Artificial intelligence (AI), 4
Asymptotic behavior, 50
Asymptotic theory, 49
Asynchronous updates, 304
Autocorrelation, 193
Autocovariance, 193
Auto-encoder, 266
Autoregressive models, 192

B

Back-propagation, 158
Backshift operator, 194
Backward quantity, 224
BARRA model, 177
Batch-mode reinforcement learning, 282

Baum-Welch algorithm, 224
Bayes' factor, 66
Bayesian data analysis, 48
Bayesian filtering, 62
Bayesian linear regression, 82
Bayesian model averaging (BMA), 69
Bayesian network, 72
Bayesian neural networks, 149
Behavioral cloning, 421
Bellman equation, 295
Bellman flow constraints, 425
Bellman iteration, 301
Bellman optimality equation, 297
Bernoulli random variable, 51
BFGS, *see* Broyden-Fletcher-Goldfarb-Shanno (BFGS)
Bias, 54
Biased data, 528
Bias-variance dilemma, *see* Bias-variance tradeoff
Bias-variance tradeoff, 112, 122, 141
Bi-directional information flow, 537
Bitcoin, *see* Cryptocurrencies
Black-Litterman (BL) model, 506
Black-Scholes-Merton (BSM), 349
Blockchain, 6, 7
BM, *see* Boltzmann machine (BM)
BMA, *see* Bayesian model averaging (BMA)
Boltzmann distribution, 429
Boltzmann machine (BM), 151
Boltzmann policy, 429
Bounded rationality, 507
Box-Jenkins, 205
Broyden-Fletcher-Goldfarb-Shanno (BFGS), 232

BSM, *see* Black-Scholes-Merton (BSM)
 BUGS, 234

C

Calibration, 232
 Call option, 349
 Causal entropy, 434
 Cayley-Hamilton theorem, 196
 Children, 229
 Coarse-grained variables, 529
 Complexity penalty, 95
 Conditionally independent, 71
 Confusion matrix, 210
 Conjugate distribution, 60
 Constant relative risk aversion utility (CRRA), 403
 Convex conjugate, 463
 Convolutional neural network, 257
 Corporate default, 526
 Correlation breakdown, 73
 Covariance kernel, 81
 CRRA, *see* Constant relative risk aversion utility (CRRA)
 Cryptocurrencies, 6
 Customer choice, 444

D

DAG, *see* Directed acyclic graph (DAG)
 Data augmentation, 234
 Data-feature map, 8
 Data mining, 8
 Decumulative distribution function, 479
 Deep learning, 149
 Defined contribution pension plan, 405
 Deterministic policies, 292
 DFT, *see* Discrete Fourier transform (DFT)
 Dickey-Fuller test, 18
 Dilated convolution, 264
 Directed acyclic graph (DAG), 234
 Discount factor, 290
 Discrete Fourier transform (DFT), 97
 Discrete mixture model, 73
 Discriminative learning, 9
 Disturbance-based reward extrapolation (D-REX), 490
 Double Q-learning, 329
 D-REX, *see* Disturbance-based reward extrapolation (D-REX)
 Dropout, 149
 Dynamic portfolio management, 348
 Dynamic programming, 8, 299

E

Earth-mover distance, 470
 Edward, 234
 ELBO, *see* Evidence lower bound (ELBO)
 Elman network, 241
 EM, *see* Expectation maximization (EM)
 Emission matrix, 225
 Emission probability, 227
 Entropy, 11
 ε -greedy policy, 316
 Equilibrium statistical mechanics, 430
 Error, 53
 Error covariance matrix, 54
 Evidence, 56, 94
 Evidence lower bound (ELBO), 152
 Expectation maximization (EM), 74
 Experience, 307
 Experience replay, 316
 Exploration-exploitation dilemma, 281

F

FAIRL, 478
 Fast Fourier transform (FFT), 96
 FCW, *see* Financial cliff walking (FCW)
 F-divergencies, 464
 Feedforward network, 14
 Fenchel conjugate, 463
 Fenchel-Legendre transform, 463
 FFT, *see* Fast Fourier transform (FFT)
 Financial cliff walking (FCW), 305
 Financial planning, 401
 Fintech, 6
 FITC, *see* Fully independent training conditional (FITC)
 Fitted Q-iteration, 328
 F-MAX, 476
 Forward KL divergence, 474
 Forward quantity, 224
 Fraud, 6
 Frequentist data analysis, 48
 F-score, 64
 F1-score, 210, 213
 Fully independent training conditional (FITC), 97
 Fundamental factor models, 177, 178

G

GAN, *see* Generative adversarial network (GAN)
 Gated recurrent unit (GRU), 222, 249
 Gaussian mixture model, 72

Gaussian process (GP), 83
 Gaussian process IRL, 484
 Gaussian process regression, 83, 91, 484
 Gaussian time-varying policies (GTVP), 396,
 408
 Generalized Autoregressive Conditional
 Heteroscedastic (GARCH), 202
 Generalized recurrent neural networks
 (GRNNs), 248
 Generative adversarial network (GAN), 467,
 468
 Generative learning, 9
 Gibbs sampler, 151, 234
 G-learning, 393, 408
 Goal based wealth management, 406, 407
 GP, *see* Gaussian process (GP)
 Greedy policies, 298
 GRNNs, *see* Generalized recurrent neural
 networks (GRNNs)
 GTVP, *see* Gaussian time-varying policies
 (GTVP)

H

Half-life, 204, 247
 Harmonic oscillator, 524
 Hedge portfolio, 350
 Heteroscedasticity, 200
 Hidden Markov model (HMM), 222
 Hidden state, 224
 Hidden variable, 73
 Hierarchical clustering, 8
 HMM, *see* Hidden Markov model (HMM)
 Human-machine interaction, 485
 Hyperparameter, 68
 Hyperprior, 68

I

IBM Watson, 29
 Ill-posed inverse problems, 420
 Imitation learning, 420
 Importance sampling, 228, 231
 Inducing point method, 96
 Inducing points, 484
 Infinite horizon MDP, 296
 Initial probabilities, 225
 Instantons, 526
 Interaction effects, 170
 Inverse optimal control, 423
 Inverse optimization, 506
 Inverse reinforcement learning (IRL), 422
 IQR, *see* Iterative quadratic regulator (IQR)
 IRL, *see* Inverse reinforcement learning (IRL)

IRL from failure, 486
 Iterative quadratic regulator (IQR), 401

J

Jensen-Shannon divergence, 464

K

Kalman filter, 227
 Kernel interpolator, 91
 Kernel learning, 91
 KL divergence, *see* Kullback-Leibler (KL)
 divergence
 K-means clustering, 8
 Knightian uncertainty, 149
 Kolmogorov-Smirnov test, 75
 Kramers escape problem, 527
 Kriging, 92
 Kullback-Leibler (KL) divergence, 149

L

Langevin equation, 523
 Laplace's principle of indifference, 57
 Latent state, *see* Hidden state
 Latent variable, *see* Hidden variable
 Learning from demonstrations, 420
 Least squares GAN (LS-GAN), 471
 Least squares policy iteration (LSPI), 332
 Legendre-Fenchel transform, 392
 Leverage, 230
 Leverage effect, 230
 Likelihood, 51
 Likelihood function, 56, 232
 Linear architectures, 326
 Linear quadratic regulator (LQR), 396, 499,
 509
 Log-likelihood, 52
 Log-likelihood trick, 310
 Log-variance, 230
 Long short term memory (LSTM), 222, 254
 Loss function, 54
 LQR, *see* Linear quadratic regulator (LQR)
 LSPI, *see* Least squares policy iteration (LSPI)

M

Machine learning (ML), 8
 Marginal likelihood, 50, 232
 Marginal likelihood function, 65
 Market contagion, 73
 Market impact, 348, 400
 Markov chain Monte Carlo (MCMC), 149,
 230, 233

- Markov decision process, 289
 Markov network, 72
 Markov random field, 72
 Markov transition kernel, 228
 Massively scalable Gaussian process (MSGP), 96
 Matern kernel (MK), 94
 Matrix Product State (MPS) decomposition, 532
 Maximum a posteriori (MAP), 69
 Maximum Causal Entropy, 435
 Maximum Entropy (MaxEnt) IRL, 439
 Maximum entropy principle, 430
 Maximum entropy RL, 342
 Maximum likelihood estimate, 52
 Maximum likelihood estimator (MLE), 232
 MCMC, *see* Markov chain Monte Carlo (MCMC)
 MDP for option pricing, 360
 Mean absolute error, 55
 Mean squared error (MSE), 55
 Merton consumption problem, 401
 Mesh-free GPs, 101
 Metropolis algorithm, 234
 Metropolis-Hastings algorithm, 234
 Metropolis-Hastings-Green algorithm, 234
 Minimum mean squared error (MMSE), 55
 Mixture models, 72
 ML, *see* Machine learning (ML)
 MLE, *see* Maximum likelihood estimator (MLE)
 MMSE, *see* Minimum mean squared error (MMSE)
 Model evidence, 66
 Model fit, 95
 Model selection, 65
 Model weight, 69
 Monte Carlo reinforcement learning methods, 307
 MPS decomposition, *see* Matrix Product State (MPS) decomposition
 MSE, *see* Mean squared error (MSE)
 MSGP, *see* Massively scalable Gaussian process (MSGP)
 Multi-GPs, 103
 Multi-kernel, 94
 Multinomial resampling, 228, 229
- N**
 Naive Bayes' classifier, 71
 Nash equilibrium, 393
 Natural language processing, 4
- Nestov's momentum, 158
 Neural network, 152
 Non-equilibrium thermodynamics, 534
 Non-expansion operators, 329
 Non-linear architectures, 327
 Non-linear dynamics, 395, 401
 Non-perturbative phenomena, 528
 Normalized weights, 229
- O**
 Observation, 224
 Observation space, 224
 Occam's razor, 69
 Occupancy measure, 425
 Off-policy algorithms, 308, 313
 One-hot encoding, 9
 Online-learning, 62
 Optimal stock execution, 285
 Overestimation bias, 329
- P**
 Parameter expansion MCMC (PX-MCMC), 151
 Partial autocorrelation, 197
 Partially observable Markov decision process (POMDP), 293
 Particle, 228
 Particle filtering, 227, 228, 230
 Partition function, 430
 Perception-action cycle, 520, 536, 537
 Planning, 281
 Point estimate, 53
 Point estimation, 49, 53
 Policy, 291
 Policy-based reinforcement learning, 309
 Policy evaluation, 301
 Policy function, 286
 Policy iteration, 302
 POMDP, *see* Partially observable Markov decision process (POMDP)
 Pooling, 263
 Posterior distribution, 49
 Posterior model probability, 65
 Posterior odds, 67
 Posterior odds ratio, 66
 Posterior sampling, *see* Thompson sampling
 Predicting events, 210
 Prediction, 210
 Predictron, 539
 Preference-based IRL, 488
 Preference learning, 487
 Principal component analysis (PCA), 214

- Principle of insufficient reason, *see* Laplace's principle of indifference
- Prior, 48
- Probabilistic graphical models, 70
- Put option, 349
- PyMC3, 234
- Q**
- QLBS, *see* Q-learning for the Black-Scholes (QLBS) problem
- Q-learning, 315
- Q-learning for the Black-Scholes (QLBS) problem, 380, 497
- Quasi-Newton methods, 233
- R**
- Radial basis functions (RBFs), 93, 325
- Random sample, 53
- Rare events, 528
- Rashomon effect, 64
- RBFs, *see* Radial basis function (RBFs)
- RBM, *see* Restricted Boltzmann machine (RBM)
- Receiver Operating Characteristic (ROC), 212
- Receiver Operating Characteristic (ROC) curve, 210
- Recurrent neural network, 222, 240
- REINFORCE, 310
- Reinforcement learning, 8, 280
- Renormalization group (RG), 529–532
- Reparameterization trick, 152
- Replication factor, 229
- Reproducing kernel Hilbert space (RKHS), 94
- Resampling, 228, 231
- Residual sum of squares (RSS), 63
- Responsibility, 75
- Restricted Boltzmann machine (RBM), 9, 72
- Returns in reinforcement learning, 293
- Reverse KL divergence, 474
- Reward function, 284
- Reward shaping, 427
- RG, *see* Renormalization group (RG)
- RG flow equations, 530
- Risk-sensitive GAIL (RS-GAIL), 480
- RKHS, *see* Reproducing kernel Hilbert space (RKHS)
- RMSE, *see* Root mean squared error (RMSE)
- RMSProp, 158
- Robbins-Monro algorithm, 317
- Robo-advisors, 6
- Robo-advisory, 485
- Root mean squared error (RMSE), 55
- RS-GAIL, *see* Risk-sensitive GAIL (RS-GAIL)
- RSS, *see* Residual sum of squares (RSS)
- S**
- Sampling uncertainty, 50
- SARSA, 314
- Selection, 228, 231
- Self-financing portfolios, 401
- Self-organizing maps, 8
- Semi-affine function, 14
- Sequential Bayesian update, 62
- Sequential Importance Resampling (SIR), 228
- SGD, *see* Stochastic gradient descent (SGD)
- SIR, *see* Sequential Importance Resampling (SIR)
- SKI, *see* Structured kernel interpolation (SKI)
- Softmax policy, 310
- Soft Q-learning, 437
- SoR, *see* Subset of regression (SoR)
- Spline interpolator, 91
- Squared error, 54
- Stability, 195
- Stan, 234
- Standard error, 49
- State space, 224
- State-space model, 221
- State-value function, 294
- Stationarity, 18, 193, 195, 246
- Stationary product kernel, 97
- Statistical ensemble, 430
- Statistical mechanics, 432
- Statistical risk, 55
- Stochastic gradient descent (SGD), 95, 142, 152
- Stochastic policies, 292
- Stochastic process, 192
- Stochastic shortest path (SSP) problems, 299
- Stochastic volatility, 230
- Stochastic volatility with leverage (SVL), 230
- Stochastic volatility with leverage and jumps (SVLJ), 230
- Structural models, 444
- Structural models for consumer choice, 444
- Structured kernel interpolation (SKI), 97
- Subjective probability, 48
- Subset of regression (SoR), 97
- Supervised learning, 8
- SVL, *see* Stochastic volatility with leverage (SVL)
- SVLJ, *see* Stochastic volatility with leverage and jumps (SVLJ)
- Synchronous updates, 304

T

- Target network, 336
TD error, 312
Temporal difference learning, 311
Tensor decompositions, 531
Tensor train decompositions, 532
Thompson sampling, 283
Tikhonov regularization, 91
Time series, 193
Time series cross-validation, 213
Time-stationary MDP, 299
Trajectory-ranked reward extrapolation
(T-REX), 488, 490
Transition matrix, 225
Transition probability, 225
T-REX, *see* Trajectory-ranked reward
extrapolation (T-REX)
True value, 54
T-TEX, 490

U

- UGM, *see* Undirected graphical model (UGM)
Unbiased, 54
Uncertainty quantification, 150

- Undirected graphical model (UGM), 72
Uninformative prior, 57
Unobservable, 233
Unsupervised learning, 8
Update a prior, 50

V

- Value-at-risk, 73
Value-based reinforcement learning, 306
Value function, 286
Value iteration, 303
Variational inference, 149
VC-dimension, 121
Viterbi algorithm, 224, 226
Volatility shock, 73
Von Neumann–Morgenstern conditions, 358

W

- Wealth management, 401, 407
Weighted moving average smoothers, 258
White noise, 193
WinBUGS, 234
Wold decomposition, 201