Essays on the Applications of Machine Learning in Financial Markets

Muye Wang

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2021

© 2021

Muye Wang

All Rights Reserved

Abstract

Essays on the Applications of Machine Learning in Financial Markets

Muye Wang

We consider the problems commonly encountered in asset management such as optimal execution, portfolio construction, and trading strategy implementation. These problems are generally difficult in practice, in large part due to the uncertainties in financial markets. In this thesis, we develop data-driven approaches via machine learning to better address these problems and improve decision making in financial markets.

Machine learning refers to a class of statistical methods that capture patterns in data. Conventional methods, such as regression, have been widely used in finance for many decades. In some cases, these methods have become important building blocks for many fundamental theories in empirical financial studies. However, newer methods such as tree-based models and neural networks remain elusive in financial literature, and their usabilities in finance are still poorly understood. The objective of this thesis is to understand the various tradeoffs these newer machine learning methods bring, and to what extent they can improve a market participant's utility.

In the first part of this thesis, we consider the decision between the use of market orders and limit orders. This is an important question in practical optimal trading problems. A key ingredient in making this decision is understanding the uncertainty of the execution of a limit order, that is, the fill probability or the probability that an order will be executed within a certain time horizon. Equivalently, one can estimate the distribution of the time-to-fill. We propose a data-driven ap-

proach based on a recurrent neural network to estimate the distribution of time-to-fill for a limit order conditional on the current market conditions. Using a historical data set, we demonstrate the superiority of this approach to several benchmark techniques. This approach also leads to significant cost reduction while implementing a trading strategy in a prototypical trading problem.

In the second part of the thesis, we formulate a high-frequency optimal execution problem as an optimal stopping problem. Through reinforcement learning, we develop a data-driven approach that incorporates price predictabilities and limit order book dynamics. A deep neural network is used to represent continuation values. Our approach outperforms benchmark methods including a supervised learning method based on price prediction. With a historic NASDAQ ITCH data set, we empirically demonstrate a significant cost reduction. Various tradeoffs between Temporal Difference learning and Monte Carlo method are also discussed. Another interesting insight is the existence of a certain universality across stocks — the patterns learned from trading one stock can be generalized to another stock.

In the last part of the thesis, we consider the problem of estimating the covariance matrix of high-dimensional asset return. One of the conventional methods is through the use of linear factor models and their principal component analysis estimation. In this chapter, we generalize linear factor models to a general framework of nonlinear factor models using variational autoencoders. We show that linear factor models are equivalent to a class of linear variational autoencoders. Furthermore, nonlinear variational autoencoders can be viewed as an extension to linear factor models by relaxing the linearity assumption. An application of covariance estimation is to construct minimum variance portfolio. Through numerical experiments, we demonstrate that variational autoencoder improves upon linear factor models and leads to a more superior minimum variance portfolio.

Table of Contents

List of Tables vi

List of Figures vii

Acknowledgments viii

Chapter 1: Introduction 1

1.1 A Deep Learning Approach to Estimating Fill Probabilities in a Limit Order Book . 2

1.2 A Reinforcement Learning Approach to Optimal Execution 4

1.3 Variational Autoencoder for Risk Estimation 6

Chapter 2: A Deep Learning Approach to Estimating Fill Probabilities in a Limit Order Book 10

2.1 Introduction 10

2.1.1 Organization of Chapter 11

2.2 Limit Order Book and Motivation 12

2.2.1 Limit Order Book Mechanics 12

2.2.2 Implementation Shortfall: A Tradeoff Between Market Orders and Limit Orders 14

2.3 Recurrent Neural Networks 16

2.3.1 LSTM Architecture 16

2.3.2 Input Features 17

2.3.3 Hazard Rates Outputs . . . 18
2.4 Model Estimation . . . 19
2.4.1 NASDAQ ITCH . . . 20
2.4.2 Synthetic Limit Orders . . . 21
2.4.3 Maximum Likelihood Estimation . . . 23
2.5 Numerical Experiment Setup . . . 24
2.5.1 Stock Selection and Experiment Setup . . . 24
2.5.2 Execution Statistics . . . 25
2.5.3 Predictable Patterns . . . 26
2.6 Numerical Experiment Results . . . 29
2.6.1 Benchmark Models . . . 29
2.6.2 Fill Probability . . . 30
2.6.3 Conditional Expectation of Fill Time . . . 31
2.6.4 Implementation Shortfall . . . 31
2.7 Conclusion . . . 34
Chapter 3: A Reinforcement Learning Approach to Optimal Execution . . . 36
3.1 Introduction . . . 36
3.1.1 Organization of the chapter . . . 38
3.2 Limit Order Book and Optimal Stopping Formulation . . . 38
3.2.1 Limit Order Book Mechanics . . . 38
3.2.2 Price Predictability . . . 39
3.2.3 Optimal Stopping Formulation . . . 40

3.3 Supervised Learning Approach 42
3.3.1 Price Trajectory Prediction 42
3.3.2 Supervised Learning Method 42
3.3.3 Execution Policy 43
3.3.4 Temporal Difference Learning 44
3.3.5 Algorithm 48
3.3.6 Insufficiency 49
3.4 Reinforcement Learning Approach 50
3.4.1 Continuation Value 51
3.4.2 Learning Task 53
3.4.3 Temporal Difference Learning 55
3.4.4 Algorithm 56
3.4.5 Discussion 57
3.5 Numerical Experiment: Setup 59
3.5.1 NASDAQ Data Source 59
3.5.2 Experiment Setup 60
3.5.3 State Variables and Rewards 61
3.6 Numerical Experiment: Results 63
3.6.1 Best Performances 63
3.6.2 Comparative Results 63
3.6.3 Universality 69
3.6.4 Result Summary 70

Chapter 4: Variational Autoencoder for Risk Estimation 72
4.1 Introduction 72
4.2 Application: Minimum Variance Portfolio 74
4.3 Linear Factor Model 75
4.3.1 Maximum Likelihood Estimation 75
4.3.2 Linear Factor Model 76
4.3.3 Principal Component Analysis 77
4.4 Variational Autoencoders 78
4.4.1 Latent Variable Models 78
4.4.2 Variational Autoencoder 80
4.4.3 Estimation via Evidence Lower Bound 80
4.4.4 Covariance Estimation 82
4.4.5 Linear Factor Models as Variational Autoencoders 82
4.5 Numerical Experiments 84
4.5.1 Stock Return Data 85
4.5.2 Estimation Accuracy Comparison 86
4.5.3 Minimum Variance Portfolios 88
4.6 Conclusion 91
References 95
Appendix A: Numerical Results for Chapter 3 96
A.1 Summary Statistics of Selected Stocks 96
A.2 State Variables 98

A.3 Algorithm Performances 98

Appendix B: Proofs for Chapter 4 101

B.1 Lemma 4.4.1: 101

B.2 Theorem 4.4.3 103

List of Tables

2.1 Event Message 20

2.2 Price Table 20

2.3 Volume Table 20

2.4 Stock Selection Summary Statistics 25

2.5 Descriptive Statistics 25

2.6 Fill Probability Prediction Results 30

2.7 Conditional Expected Fill-Time Prediction Results 31

2.8 Implementation Shortfall 33

2.9 Average Clean-up Cost (ticks) Conditioned on Clean-up 34

3.1 Event Message 60

3.2 Price Table 60

3.3 Volume Table 60

3.4 Performance Comparison 1 63

3.5 Performance Comparison 2 70

4.1 Realized Volatility (Standardized Return) 89

4.2 Realized Volatility (Raw Return) 90

A.1 Stock Selection - Summary Statistics 98

A.2 Variables in States 98

A.3 Detailed Algorithm Performance 99

List of Figures

2.1 Limit Order Book Dynamics 13

2.2 LSTM Recurrent Neural Network 17

2.3 Hazard Rate Estimation 19

2.4 Synthetic Limit Order Simulation 23

2.5 Time of Day Pattern (h = 1 min) 26

2.6 Queue Imbalance Patterns (h = 1 min) 28

2.7 Queue Imbalance Histograms 29

3.1 Limit Order Book Dyanmics 39

3.2 Recurrent Neural Network (RNN) Architecture 62

3.3 Price Gain vs. Gradient Step 64

3.4 Average Gradient Time 65

3.5 Price Gain vs. Running Time 66

3.6 Price Gain vs. Data Accessed 67

3.7 Bias–Variance Tradeoff 68

3.8 Left: In-Sample RMSE; Right: Out-of-Sample RMSE. 69

4.1 Covariance Matrix Estimate Comparison 86

4.2 Number of Factors 88

Acknowledgements

This thesis is the result of collaborative work with my advisors, Professor Ciamac C. Moallemi and Professor Costis Maglaras. It is my great fortune to have the opportunity to study under their guidance.

I have benefited greatly from my close interactions with Professor Ciamac C. Moallemi. His talent and intellectual prowess have served as a constant source of motivation. His expertise on the subject has fundamentally shaped this thesis. Professor Costis Maglaras has been a thoughtful teacher and provided constant support and encouragement. I am deeply grateful for their support and mentorship.

I would also like to thank Professors Paul Glasserman and Daniel Russo who have been generous with their time and effort, Professors Carri Chan and Yash Kanoria for their guidance through the PhD program, and Professor Daniel Guetta who has offered me support and generosity.

I am fortunate to be part of the community at Columbia Business School. I want to thank Elizabeth Elam and Dan Spacher from the PhD office, Razvan Popescu and Benny Chang from the research computing group, Clara Magram, Winnie Leung, Maria Micheles, and Cristina Melo-Moya from the DRO division. The broader Columbia community, the residential service, the health center, and the facility service have showed tremendous leadership and commitment in helping us navigating through the COVID-19 crisis in the past year.

I have also benefited greatly from many discussions and interactions with fellow students. Seungki Min has been my teaching assistant, officemate and collaborator during my time at Columbia.

He has been a generous mentor and a kind friend. The experience that we shared will forever be a highlight of these past years.

I am thankful for Yiwen Shen, Sharon Huang, Pu He, Jiaqi Lu, and Pengyu Qian, among others. I also wish to acknowledge Steven Yin and James Yang for their friendship and support. I am grateful to Sasha Chen for her patience, kindness, and companionship.

Finally, I would like to thank my parents, Yuguang Wang and Wanyue Qian, for their unconditional love and support. It is from them that I inherent a thirst for knowledge which has lead to my pursuit of a PhD. Their passion for their work has always been a source of inspiration. I am also grateful to Jinghua Qian, Nick, and Bobby for their love and support. I am forever indebted to my family, and it is to them that I dedicate this thesis.

Chapter 1: Introduction

Over the past two decades, machine learning has enjoyed enormous empirical success which has led to its widespread adoption in many industries. This adoption has been so profound and impactful that Andrew Ng said the following in a talk titled "Artificial Intelligence is the New Electricity."[1]

> *"Just as electricity transformed industry after industry almost 100 years ago, today I think AI will do the same."* — *Andrew Ng*

Despite the widespread success of machine learning, its adoption in financial markets remain somewhat elusive due to certain unique challenges. Firstly, data from financial markets has low signal-to-noise ratio. This makes it difficult for machine learning to distinguish signal from noise. In other words, models are prone to overfit. In order to combat this, machine learning algorithms need to be tuned meticulously using methods such as cross-validation. Secondly, financial markets aren't static. Rather, they evolve over time. For example, merely two decades ago, most stock exchanges in the U.S. were operated by human traders in trading floors. Nowadays, most stock exchanges are electronic. As a result of changes like this, the markets from decades ago aren't good reflections of what the markets are like today. More specifically, historical market data from the distant past can't accurately reflect the market conditions now and certainly aren't able to predict future market accurately. Therefore, from a data scientist's perspective, in order to predict future market dynamics, the historical data that can be used is limited only to the recent past. This limitation of data presents series of challenges in machine learning, which we will discuss in more detail in Chapter 4. Lastly, the challenge that is perhaps most unique to financial markets, is that the markets have a certain self-correcting mechanism. Because practitioners who make

[1] See Stanford MSx Future Forum. January 25, 2017.

predictions in the markets are often market participants themselves, once they discover predictable signals, they can profit from them directly in most cases. This in turn diminishes the strength of the signal or even eliminates them entirely. This is also referred to as "alpha decay." As a result of this mechanism, financial markets are largely efficient — predicting future prices are incredibly difficult and market anomalies are often very subtle and elusive.

Despite these challenges, we present a few areas in finance where machine learning can bring substantial benefits. In Chapter 2, we use deep learning to predict the execution outcomes of limit orders. This improves trading implementation when the choice of market orders and limit orders plays an important role. In Chapter 3, we formulate an optimal execution problem through reinforcement learning. By incorporating price predictabilities and limit order book dynamics, reinforcement learning outperforms many benchmark methods including a supervised learning method based on price prediction. Chapter 4 considers the problem of estimating asset return covaraince. We discuss a few estimation methods including linear factor models and variational autoencoders, and conduct numerical experiments to demonstrate their performances.

The rest of this chapter introduces these three following chapters in depth by providing backgrounds and relevant literature.

1.1 A Deep Learning Approach to Estimating Fill Probabilities in a Limit Order Book

Most modern financial exchanges use electronic limit order books (LOBs) as a centralized system to trade and track orders. In such exchanges, resting limit orders await matching to contra-side market orders.[2]

Because exchanges typically offer multiple order types, when traders submit an order, they often face the choices of many order types. The most common choice is between a market order and a limit order. Market orders are orders that execute immediately at the best current price. Such orders are employed by traders whose priority is immediate executions. Limit orders are

[2]A market order executes immediately at the current best price. A marketable limit order specifies a limit price as a constraint, but that constraint is not binding and the order executes immediately. For the purposes of our study, we use these two terms interchangeably.

orders that execute only at a specified price or better. As a result, limit orders typically don't execute right away, and in some cases, limit orders don't execute at all. The delay between a limit order's submission and its execution is called the "time-to-fill" or "fill time." In order to choose between market orders and limit orders intelligently, it's important for a trader to understand the uncertainty of limit order executions, more specifically, the fill probability with a given time horizon, or equivalently, the distribution of the fill times.

Deep learning is a branch of machine learning that uses neural networks to capture intricate patterns in data. A typical neural network consists of layers of artificial neurons that use activation functions to model nonlinear relationships. A deep neural network with multiple layers effectively represents the function composition of all the layers of these activation functions. As a result, a deep neural network can represent very complicated functions and capture patterns that traditional linear models can not.

In Chapter 2, we apply deep learning technology to study the uncertainty of the limit order executions, that is, the fill probability or equivalently the distribution of the fill times. Accurate fill probability predictions help traders better decide between market orders and limit orders, which we demonstrate via a prototypical trading problem.

The study of limit order books dates back to the late 1980s. The following is not a comprehensive review, but rather a highlight of a few notable studies that are most relevant to our work in Chapter 2. Angel (1994) derives an analytical expression for limit order fill probability, conditional upon an investor's information set. However these results are derived under some rather strong assumptions. Hollifield, Miller, and Sandas (2004) build a structural model of a limit order book market and characterize the tradeoff between market orders and limit orders. They compute a semi-parametric estimator of the model primitives using data from the Stockholm Stock Exchange. Another study that compares the use of market orders and limit orders is that of Petersen and Fialkowski (1994). They conduct an empirical study using data from the NYSE and report a significant difference between the posted spread and the effective spread paid by investors. Lo, MacKinlay, and Zhang (2002) develop an econometric model to estimate time-to-first-fill and

time-to-completion. They find that execution times are very sensitive to the limit price, but not as sensitive to the order size. They also find that many hypothetical limit order execution models are very poor proxies for actual limit order executions. Cho and Nelling (2000) conduct an empirical study and report that the longer a limit order is outstanding, the less likely it is to be executed.

With the rise of big data and machine learning, researchers have started to apply machine learning to the study of finance. Heaton, Polson, and Witte (2016) outlines the general frameworks of deep learning and many areas in finance that deep learning ideas can be useful. Some more specific deep learning applications include the study of Xiong, Nichols, and Shen (2015), Carr, Wu, and Zhang (2019), and Ban, Karoui, and Lim (2018).

Because deep learning typically requires large data sets, its applications in high frequency domain have also shown to be promising. Sirignano and Cont (2019) use a recurrent neural network to predict next immediate price changes and further argue that there are certain universality in the price formation process across stocks. Zhang, Zohren, and Roberts (2019) train a deep learning model to predict price movements in the near future. Dixon, Klabjan, and Bang (2017) use a neural network to predict financial market movement directions and demonstrate its application in a simple trading strategy. Other machine learning applications in this area include the work of Tran et al. (2017), Tran et al. (2019), Tsantekidis et al. (2017), Passalis et al. (2018) and Ntakaris et al. (2018).

1.2 A Reinforcement Learning Approach to Optimal Execution

Algorithmic execution is an important part of asset management, and it involves many practical considerations. Traders who are looking to fulfill an order have many choices of executions. They can fulfill their orders on exchanges, which typically reveals the particular trade in real time to the public. Alternatively, they can also trade on a dark pool, which provides certain anonymity. The size of the order matters in the execution as well. A large order might deplete the liquidity at the best price, incurring significant transaction cost. To combat this, the trader has an incentive to divide the large order into smaller child orders and trade gradually over time. For smaller orders,

the timing aspect becomes more important. If a trader can predict future price movements, placing the order at the right time could reduce implementation shortfall significantly. Different utility functions lead to different execution algorithms as well. Some utility function only incorporates the average implementation shortfall, whereas others take variance of the execution price into account. Maximizing different utility functions leads to different execution algorithms.

In financial literature, the problem of optimal execution aims to balance various tradeoffs while optimize a specific utility function. One of such tradeoffs is between the immediate transaction cost and the price uncertainty in time. Given a large order to execution, if a trader trades all shares in a single execution, the liquidity at the best price will be depleted, incurring significant transaction cost. By trading smaller orders gradually over time, the trader reduces the transaction cost, but inevitably prolongs the execution horizon. This exposes the trader to a greater degree of price uncertainty. To execute optimally, the trader needs to balance these two conflicting interests. There is a large body of literature that formalizes this heuristic, which we will introduce briefly below.

The majority of the work in this area uses model-based approaches. These approaches impose models on price dynamics, market impact, or other aspects of the execution. These theoretical models advance our mathematical understanding of the various tradeoffs in execution. However, in order to ensure tractability, these models impose simplifying assumptions deviating them from reflecting the reality. In Chapter 3, we aim to develop a data-driven approach from real market data to model the dynamics in trading execution. Earlier work in the area of optimal execution problem includes Almgren and Chriss (2000) and Bertsimas and Lo (1998). These two papers lay the theoretical foundations for many further studies, including Coggins, Blazejewski, and Aitken (2003), Obizhaeva and Wang (2013), and El-Yaniv et al. (2001).

The paper that is perhaps most closely related to our work is Nevmyvaka, Feng, and Kearns (2006). They also apply reinforcement learning (RL) to the problem of optimal execution, but there are also many differences. They consider the dividing problem of the parent order and the goal is to obtain an optimal trading schedule, whereas we apple RL to solve the child order problem using a

single execution. On a more technical aspect, they use a tabular representation to present the state variables, which force the state variables to be discretized. We allow continuous state variables by utilizing neural networks. Other differences include the action space, feature selections, and numerical experiment as well.

Another area in finance where optimal stopping is an important practical problem is pricing American options. Motivated by this application, Longstaff and Schwartz (2001) and Tsitsiklis and Van Roy (2001) have proposed using regression to estimate the value of continuation and thus to solve optimal stopping problems. Similarly to this work, at each time instance, the value of continuation is compared to the value of stopping, and the optimal action is the action with the higher value. The regression-based approach is also different in a number of ways. One difference is the choice of model. They use regression with linear model to estimate continuation values where as we use nonlinear neural networks. Another difference is that they fit a separate model for each time horizon using a backward induction process, which increases the remaining horizon one step at a time. By contrast, we fit a single neural network for all time horizons. Our approach can learn and extrapolate features across time horizons. This also leads to a straightforward formulation of temporal difference learning, which we will discuss in Section 3.3.4 and Section 3.4.3.

This work also joins the growing community of studies applying machine learning to tackle problems in financial markets. Sirignano (2019) uses neural networks to predict the direction of the next immediate price change and also reports the similar universality among stocks. Kim, Shelton, and Poggio (2002) utilize RL to learn profitable market-making strategies in a dynamic model. Park and Van Roy (2015) propose a method of simultaneous execution and learning for the purpose of optimal execution.

1.3 Variational Autoencoder for Risk Estimation

In portfolio theory, the covariance of asset returns plays an important role in risk management as well as portfolio construction. More specifically, let $x \in \mathbb{R}^n$ be a random vector representing the return of n investable assets. Let $\mu \in \mathbb{R}^n$ be its mean and $\Sigma \in \mathbb{R}^{n \times n}$, a positive semi-definite

matrix, be its covariance matrix.

The covariance matrix Σ is crucial in evaluating the volatility within an given portfolio. Let $w \in \mathbb{R}^n$ be any portfolio allocating capital within these n investable assets, the volatility of the portfolio is given by

$$\sigma_w = \left(w^\top \Sigma w\right)^{1/2}.$$

The covariance matrix Σ is also important in portfolio construction. Most famously, Markowitz (1952) proposes a framework to obtain an optimal mean-variance portfolio, where the covariance matrix Σ is an important input. Specifically, given a risk tolerance parameter $\lambda > 0$, a portfolio w is optimal if it maximizes a mean-variance utility function.

$$w^* = \underset{w}{\text{argmax}} \; \{w^\top \mu - \lambda w^\top \Sigma w\}.$$

This optimization problem is typically subject to various portfolio constraints. Most common constraints include the budget constraint $w^\top 1 = 1$, or the long-only constraint $w > 0$.

However, the covariance matrix Σ is generally unknown in practice and needs to be estimated from historical return data. When the number of assets n is large, the problem of estimating covariance matrix becomes a challenging problem. This is principally because there are many parameters that need to be estimated — the number of free parameters in matrix Σ is in the order of n^2, which scales quadratically with the number of assets n. To make the matter worse, because stock returns are time-varying, only the historical return data from the recent past can accurately reflect the covariance of stock return today. This limits the amount of data that can be used in estimation. From a statistical standpoint, when the amount of data is small and the number of parameters are large, the estimation accuracy typically suffers.

One way to alleviate this problem is to impose structure on the covariance matrix and reduce the number of free parameters. In modeling stock returns, a common choice is to impose a factor structure that uses a smaller number of variables to explain the variations in cross-sectional stock returns. A class of models that achieves this is linear factor models. A linear factor model connect

a low-dimensional vector, $z \in \mathbb{R}^k$, to the higher-dimensional stock return vector x through a linear transformation. This is given by

$$x = Lz + \epsilon,$$

where $L \in \mathbb{R}^{n \times k}$ is the matrix representing the linear transformation and $\epsilon \in \mathbb{R}^n$ is the residual noise. Under the isotropic Gaussian setting where $z \sim N(0, I_k)$ and $\epsilon \sim N(0, \sigma^2 I_n)$, the distribution of x is given by

$$x \sim N(0, LL^\top + \sigma^2 I_n). \tag{1.1}$$

Equation (1.1) represents the structure that the linear factor model imposes on the distribution of stock returns x. Now instead of estimating the covariance matrix Σ directly, we just need to estimate the model parameters L and σ, which will lead to a covariance matrix estimate as $\hat{\Sigma} = \hat{L}\hat{L} + \hat{\sigma}^2 I_n$. More details about linear factor models and their estimation procedures are discussed in Chapter 4.

In linear factor models, the factors and the stock returns are related linearly. This leads to certain restrictions on the distribution of the observable variables. Specifically, the distribution of the stock return has to follow a Gaussian distribution, as in (1.1). Variational autoencoders (VAEs) are a class of latent variables models, and they can be used to relax the linear assumption, consequently, relax the Gaussian restriction.

Estimating the asset return covariance has long been a problem, especially when the number of asset is large. Various covariance structure has been proposed in the literature. Ledoit and Wolf (2001) propose a shrinkage method by optimally weighted averaging two existing estimators: the sample covariance matrix and single-index covariance matrix. This is a different way of regularizing the covariance matrix estimate without specifying a multi-factor structure. Another paper on the same topic is Ledoit and Wolf (2003). Autoencoder has also been used in modeling financial data in the literature. Gu, Kelly, and Xiu (2019) and Suimon et al. (2020) are two such examples.

In Chapter 4, we make connections between linear factor models and VAEs, and demonstrate how VAEs can be applied to estimate stock return covariances. As an application of covariance

matrix estimation, we also demonstrate the economic value of various estimates by constructing minimum variance portfolios.

Chapter 2: A Deep Learning Approach to Estimating Fill Probabilities in a Limit Order Book

2.1 Introduction

Most stock exchanges offer multiple order types. This presents a world of possibilities for trading implementation. More specifically, when traders want to buy or sell a certain amount of stocks, they need to choose an order type that best meets their requirements. The most common types of orders are market orders and limit orders. Market orders execute immediately at the current best available price, whereas limit orders only executes at specified price or better. As a result, limit orders don't execute right away, in some cases, they don't execute at all. Due to the price specification, limit orders can capture a possible price premium over market orders — a limit order can be submitted at a better price than the current best available price and gets executed at a future time. However, because limit orders aren't guaranteed to be executed, this price premium is only realized with certainly probability — the fill probability of limit orders.

In order to best choose between market orders and limit orders, it's important to for traders to understand the uncertainty of limit order executions. In other words, the fill probabilities within a certain time horizon. In this chapter, we develop a data-driven approach to estimate the uncertainty of limit order executions, and demonstrate its economic utilities in trading implementation through numerical experiments.

The main contributions of this chapter are as follows.

We propose a data-driven approach based on a specific recurrent neural network (RNN) architecture to predict limit order executions. Most studies on limit order executions use a model-based approach, which inevitably suffers from various model limitations, such as model misspecification. We propose a data-driven approach that takes advantage of the abundance of

exchange market data. In order to model the temporal dynamics of limit order executions, we construct a RNN as opposed to a more traditional feed-forward neural network. In this study, we directly estimate the distribution of the fill times by designing a hazard rate approach. As far as we know, this is the first study that directly predicts limit order executions via the distribution of fill times.

We demonstrate better prediction accuracy against benchmark models. The performance of the RNN are measured using two metrics — fill probability and expectation of the fill times conditioned on execution. We use traditional estimation methods such as logistic regression to establish benchmarks. The RNN method outperforms the benchmarks on both of the metrics over various time horizons.

We demonstrate better performance in a prototypical execution problem. Better limit order execution predictions have important implications in trading strategy implementation. We specify a benchmark trading problem that considers the tradeoff between market orders and limit orders in executing a single share, with the goal of minimizing implementation shortfall. As RNN predicts fill probabilities more accurately, it also improves the trading strategy by reducing implementation shortfall.

2.1.1 Organization of Chapter

The remainder of the chapter is organized as follows. Section 2.2 outlines the limit order book dynamics and demonstrates the tradeoff between market orders and limit orders through a trading problem. The optimal trading strategy of the problem motivates the estimation of limit order fill probabilities. Section 2.3 describes recurrent neural networks and the hazard rate method for distribution estimation. Section 2.4 describes the NASDAQ ITCH data source, the simulation procedure of generating synthetic limit orders, and the maximal likelihood estimation of the RNN. Section 2.5 lists descriptive statistics of these synthetic limit orders and demonstrates a few predictive patterns. Section 2.5 presents the prediction results. The trading problem from Section 2.2 is revisited and the economic value of better fill probability predictions is illustrated. Section 2.7 concludes

with a brief overview and some remarks regarding the limitations of this work.

2.2 Limit Order Book and Motivation

In this section, we will introduce the mechanics of limit order books and discuss a prototypical trading problem that considers the tradeoff between limit orders and market orders. The optimal trading strategy requires fill probability as an input, which motivates the fill probability estimation problem.

2.2.1 Limit Order Book Mechanics

Limit order books are responsible for keeping track of all resting limit orders at various price levels. Because investors' preferences and positions change over time, the limit order books also need to be dynamic and change over time. During trading hours, market orders and limit orders are constantly being submitted and traded. These events alter the resting limit orders, and consequently, the shape of the limit order books. Other market events that alter the shape of the limit order books include partial or complete cancellations of resting limit orders.

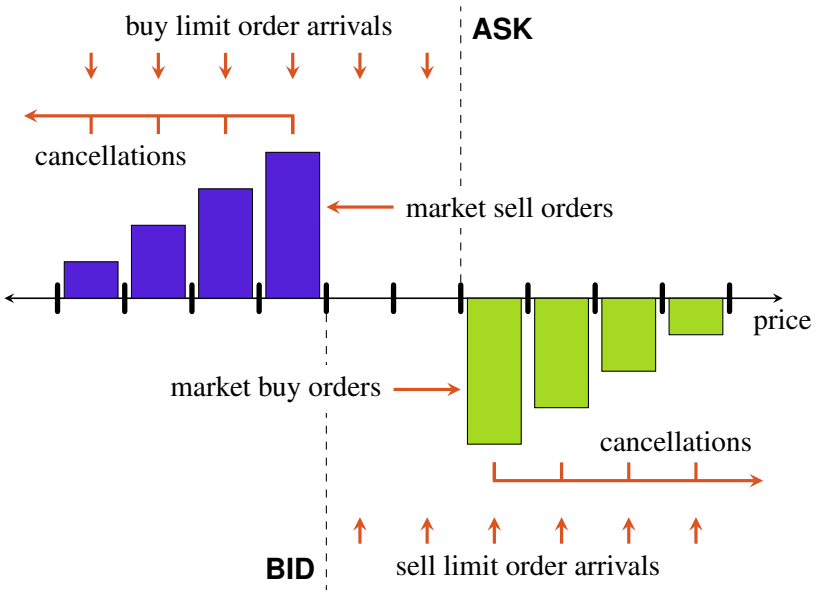


Figure 2.1: Limit orders are submitted at different price levels. The ask prices are higher than the bid prices. The difference between the lowest ask price and the highest bid price is the "bid-ask spread." Mid-price is the average of the best ask price and the best bid price.

Limit order books are paired with a matching engine that matches incoming market orders with resting limit orders to fulfill trades. The most common rule that the matching engine operates under is "price-time priority." When a new market order is submitted to buy, sell limit orders at the lowest ask price will be executed; when a new market order is submitted to sell, buy limit orders at the highest bid price will be executed. For limit orders at the same price, the matching engine follows time priority — whichever order was submitted first gets executed first.

The configuration of limit order books and the matching rule prompt researchers to model limit order books as queuing systems (e.g., Cont, Stoikov, and Talreja (2010), Moallemi and Yuan (2016), Toke (2013)). Market orders correspond to service completion and limit orders correspond to customer arrival. The difficulty of these approaches lies in the complexity of the dynamics of these market events. Empirical evidence suggests that the rates of these market events change based on market conditions. For example, Biais, Hillion, and Spatt (1995) find evidence that investors are more likely to submit limit orders (rather than hitting the quotes) when the bid-ask spread is large or the order book is thin. Cho and Nelling (2000) report that the longer a limit order is outstanding,

the less likely it is to be executed.

2.2.2 Implementation Shortfall: A Tradeoff Between Market Orders and Limit Orders

The choice between market orders and limit orders can be viewed as a tradeoff between an immediate execution and a price premium. A buy market order executes at the best ask price whereas a buy limit order executes at a lower price. Therefore a limit order gains at least a bid-ask spread over a market order per share. The analogous situation holds for sell orders. However, even though limit orders offer a price premium, the execution isn't guaranteed. Therefore, the price premium is only realized with a certain probability, namely the fill probability.

To better demonstrate this tradeoff, consider the following stylized trading problem. Suppose an agent seeks to buy a share of stock over a fixed time horizon $[0, h]$. (The selling problem is analogous.) The agent seeks to minimize the implementation shortfall

$$\text{IS} = \mathbb{E}[p_E - p_M(0)],$$

where p_E is the execution price which could be a random variable, and $p_M(0)$ is the mid-price at the arrival time 0. This task can be accomplished by using either a market order or a limit order. These two choices would lead to different execution outcomes as follows.

1. **Market Order:** Submit a market order at the arrival time 0 and pay the current best ask price. This leads to an implementation shortfall of

$$\text{IS}_{\text{mkt}} = p_A(0) - p_M(0).$$

2. **Limit Order:** Submit a limit order at the best bid price at the arrival time 0. If it is not filled by time h, place a "clean-up trade" with a market order at time h. The clean-up cost can be expressed as

$$\text{C}_{\text{clean-up}} = p_A(h) - p_M(0).$$

Let T be a random variable that denotes the fill time for this limit order. Then the expected implementation shortfall is

$$\mathbb{E}[\mathrm{IS}_{\text{limit}}] = p(T \le h)[p_B(0) - p_M(0)] + p(T > h)\mathbb{E}[\mathrm{C}_{\text{clean-up}}|T > h].$$

For simplicity, assume that the bid-ask spread stays constant and is equal to S in this time horizon, namely

$$p_A(0) - p_M(0) = S/2, \quad p_B(0) - p_M(0) = -S/2.$$

The constant-spread assumption simplifies the above two implementation shortfalls to

$$\mathrm{IS}_{\text{mkt}} = S/2; \qquad \mathbb{E}[\mathrm{IS}_{\text{limit}}] = p(T \le h) \cdot -S/2 + p(T > h) \cdot \mathbb{E}[\mathrm{C}_{\text{clean-up}}|T > h].$$

For a risk-neutral agent, the optimal trading strategy is to pick the order with the smaller cost. In this case, the agent would choose a limit order if and only if $\mathbb{E}[\mathrm{IS}_{\text{limit}}] < \mathrm{IS}_{\text{mkt}}$. This is equivalent to

$$p(T \le h) > \frac{\mathbb{E}[\mathrm{C}_{\text{clean-up}}|T > h] - S/2}{\mathbb{E}[\mathrm{C}_{\text{clean-up}}|T > h] + S/2} \triangleq \Theta.$$

Otherwise the agent would choose a market order. If the market is efficient (the expected future price equals the current price) and the price changes and fills are independent, then

$$\mathbb{E}[\mathrm{C}_{\text{clean-up}}|T > h] = \mathbb{E}[\mathrm{C}_{\text{clean-up}}] = \mathbb{E}[p_A(h) - p_M(0)] = p_A(0) - p_M(0) = S/2.$$

This would lead to $\Theta = 0$ and consequently a market order would always be more preferable. However, in reality, due to the adverse selection of limit order executions, if a buy limit order doesn't get filled, the price has typically moved higher by time h. Therefore, in this case,

$$\mathbb{E}[\mathrm{C}_{\text{clean-up}}|T > h] \ge S/2.$$

This makes the threshold Θ positive, and therefore a meaningful threshold on fill probability.

In practice, because the fill probability $p(T \leq h)$ is unknown, it needs to be estimated in order to implement the above trading strategy. This motivates the rest of the chapter, where we develop a RNN-based method to estimate the fill probability of particular limit orders.

2.3 Recurrent Neural Networks

For a new limit order an agent is contemplating placing, consider the problem of predicting fill probability over various time horizons. This problem can be formulated as a supervised learning problem. We collect limit orders submitted under various market conditions and record their fill times. Input features are selected to represent the market conditions and a neural network is trained to predict fill probabilities. The rest of this section outlines the input features, the recurrent neural network (RNN) architecture, and the hazard rate method used to predict limit order fill probabilities.

2.3.1 LSTM Architecture

In a limit order book market, past events can have predictive power of the immediate future. Therefore, we not only collect input features when the limit order is submitted, we also collect features prior to the order's submission. These sets of input features collectively represent the market condition and help predict limit order's fill probability.

To process time series of input features, a special neural network structure is required. Traditional feed-forward neural networks are static models and they are not equipped to model temporal dynamics in data. RNNs are a class of neural networks specially designed to model the temporal dynamics between past observations and future observations. In this study, we use a long short-term memory (LSTM) network, a specific RNN structure that helps establish long term temporal dependency.

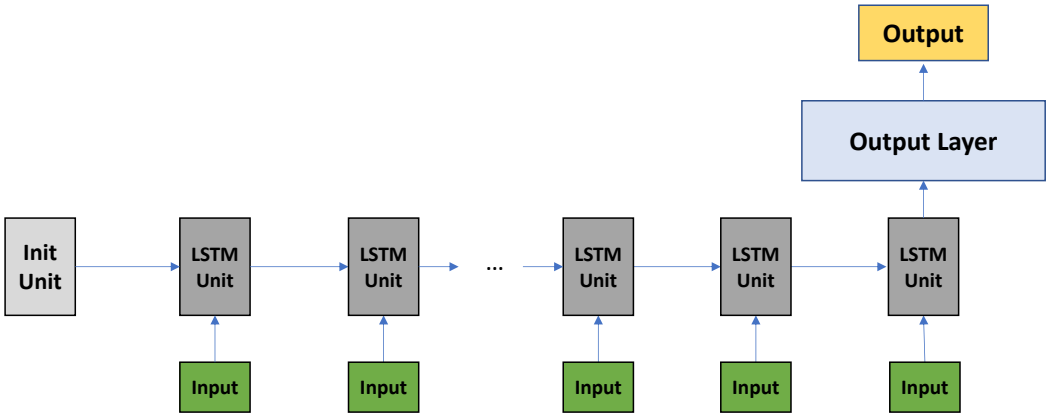


Figure 2.2: LSTM Recurrent Neural Network

Figure 2.2 depicts the RNN architecture used in this study. Each input unit represents a set of input features sampled at chronological order. LSTM units are used as hidden states to model temporal dynamics. These LSTM units connect input features from different time instances and capture any effect they have on the prediction output. The output of the neural network is a set of hazard rates, which collectively represent the distribution of the fill times. The details of inputs and outputs are described below.

2.3.2 Input Features

Each time the market is sampled, a set of input features outlined below are collected.

- **General information** in the market such as time of day, time since last trade, mid-price, and bid-ask spread are collected as features.
- **Depth of the order book** signifies the immediate supple and demand of the stock at various prices. Depth (number of resting shares) at each of the top 5 price levels on both near and far side are computed and collected as features.
- **Flow information** is included as features to capture the evolution of the order book. These include the number of shares added to the limit order book, the number of shares executed, and the number of shares cancelled since last time features are collected.

- **Intensity measures** are computed for trades and price changes and collected as features. Intensity is modeled as an exponentially decay function with increments at time instances of trades. Let S_t be the size of the trade at any given time t, then the intensity measure $X(t)$ can be modeled as

$$X(t + \Delta t) = X(t) \cdot \exp(-\Delta t / t_c) + S_{t+\Delta t},$$

  where t_c is a constant controlling the rate of the decay. Similar intensity measures can be built for price changes and other market events as well.

These input features treat the market symmetrically. In other words, buy limit orders and sell limit orders are not distinguished by the features. "Near side" is the side of the market that the limit order is submitted to — if the limit order is to buy, the bid side of the market is the near side. The "far side" is the opposite side of the market that the limit order is submitted to — if the limit order is to buy, the ask side of the market is the far side.

2.3.3 Hazard Rates Outputs

The limit order fill times can be conceptualized as a type of "survival time" from survival analysis, a branch of statistics for analyzing the duration of time until a certain event occurs, such as death in biological organisms or failure in mechanical systems. The event in our study that defines the analysis is the execution of limit orders. In these type of analyses, hazard function is commonly used to model the distribution of survival time. The simplest survival time distribution, the exponential distribution, is modeled by a constant hazard rate. In this study, in order to model the distribution of limit order fill times, we estimate constant hazard rates on multiple time intervals. This method can model distributions more general than the exponential distribution, while also maintaining tractability.

Specifically, for a particular limit order, let f_T and F_T be the density and the cumulative distribution of its fill time T. A supervised learning problem can be formulated to capture the mapping from the input features to the fill time density function.

In order to parameterize f_T concretely, we partition the time axis into a set of pre-determined intervals and estimate a constant hazard rate for each interval. More specifically, the time axis $\mathbb{R}_+$ can be partitioned into $M+1$ intervals: $[0, \tau_1), [\tau_1, \tau_2), ..., [\tau_M, +\infty)$. For each interval, a constant hazard rate can be estimated. Let $\lambda \in \mathbb{R}^{M+1}$ be the set of hazard rates on these intervals, as illustrated in Figure 2.3.

Figure 2.3: Hazard Rate Estimation on Pre-determined Intervals

This set of pre-determined intervals and the corresponding hazard rates uniquely determine a distribution on $\mathbb{R}_+$. This distribution also provides explicit expressions. For $t \in \mathbb{R}_+$, let $i^*(t) = \max\{i : \tau_i < t\}$. If $t \geq \tau_M$, then let $i^*(t) = M$. Then the cumulative hazard rate up to t can be expressed as

$$\Lambda(t) = \sum_{i=1}^{i^*(t)} \lambda_i \cdot (\tau_i - \tau_{i-1}) + \lambda_{i^*(t)+1} \cdot (t - \tau_{i^*(t)}).$$

This leads to the cumulative and the density distribution functions as follows

$$F_T(t) = 1 - e^{-\Lambda(t)}; \qquad f_T(t) = \lambda_{i^*(t)+1} \cdot e^{-\Lambda(t)}.$$

Now that the distribution of the fill times are expressed collectively as a set of hazard rates, the problem becomes estimating the hazard rates. Therefore, we design the output of the RNN to be a set of estimated hazard rates $\hat{\lambda} = (\hat{\lambda}_1, \hat{\lambda}_2, ..., \hat{\lambda}_{M+1})$, corresponding to each pre-determined time interval. This model can be estimated through maximum likelihood estimation, which we will discuss in the next section.

2.4 Model Estimation

This section outlines the estimation procedure of the RNN model. This includes discussions about the data source, the simulation approach to generate synthetic limit orders and the maximum

likelihood estimation.

2.4.1 NASDAQ ITCH

Our approach is tested using NASDAQ ITCH data (see NASDAQ 2010), which provides Level III market data. Level III market data contains message feeds for events that have transpired in the market. Common market events that change the limit order book include "Add Order", "Order Executed", "Order Cancel" and "Order replaced." These market events occur throughout the trading hours and constantly change the limit order books. A sample of an "Add Order" event message is shown in Table 2.1.

time	ticker	side	shares	price	Type
9:30:00.4704337	BAC	B	2000	12.02	"A"

Table 2.1: This event reads: A bid limit order of 2000 shares of BAC stock is added to the LOB at price level $12.02 at 9:30:00.4704337.

From these event messages, a limit order book can be constructed to compute the total number of resting shares (depth) at each price level. The limit order book is dynamic and updates each time a new market event occurs.

time	b.prc5	b.prc4	b.prc3	b.prc2	b.prc1	a.prc1	a.prc2	a.prc3	a.prc4	a.prc5
9:30:00.4704337	12.01	12.02	12.03	12.04	12.05	12.06	12.07	12.08	12.09	12.10
9:30:00.8582938	12.01	12.02	12.03	12.04	12.05	12.06	12.07	12.08	12.09	12.10

Table 2.2: The above table is a snapshot of the limit order book displaying the prices of the top 5 price levels on both sides of the market at two timestamps. The event from Table 2.1 doesn't change the prices at each level.

time	b.vol5	b.vol4	b.vol3	b.vol2	b.vol1	a.vol1	a.vol2	a.vol3	a.vol4	a.vol5
9:30:00.4704337	10000	43700	13100	12100	7500	5200	15300	15900	17000	22200
9:30:00.8582938	10000	41700	13100	12100	7500	5200	15300	15900	17000	22200

Table 2.3: The above table is a snapshot of the limit order book displaying the number of shares on the top 5 price levels on both sides of the market at two timestamps. The event from Table 2.1 reduces 2000 shares at price $12.02.

2.4.2 Synthetic Limit Orders

In order to estimate fill time of a new limit order, we need a data set of new limit orders, input features, and associated fill times that are submitted under various market conditions. One might seek to use real limit orders from the market, however there are some immediate issues:

- Censoring: Most limit orders are canceled before they are executed. This makes the fill time observations highly censored.
- Selection Bias: Informed traders may have strategies that influence the submission of limit orders. These strategies can be based on factors such as short-term price predictions. Orders such as these may have very different fill time distributions than the orders of uninformed traders. In order to predict fill times for uninformed traders, we need unbiased fill time observations of uninformed orders.

Due to these issues, we choose to simulate synthetic limit orders to generate data. These synthetic limit orders are assumed to be infinitesimal and devoid of any market impact. We randomize these order to buy and sell, and their submission times are uniformly sampled throughout the trading hours. These orders are then submitted to the best price level in the same side of the limit order book. As the limit order book evolves over time, the queue positions of these synthetic limit orders also change in the order book. We keep track of these positions and continuously check fill conditions. If the fill conditions are met, we then regard the limit order as executed; if the fill conditions are never met and the market closes, we regard the limit order as unexecuted.

Fill conditions are meant to track the progress of the limit order and identify its fill time had it actually been submitted. Fill conditions are defined as follows.

1. New Limit Order:
    - If a new buy limit order comes in at a higher price than that of a synthetic sell order, then the synthetic sell limit order is filled.

- If a new sell limit order comes in at a lower price than that of a synthetic buy order, then the synthetic buy limit order is filled.

2. New Market Order:

   - If a market order comes in at a different price as that of a synthetic limit order, then the same logic above applies.
   - If a market order comes in at the same price as a synthetic limit order, then the synthetic limit order is executed if the size of the market order is larger than the share of limit orders in front of the synthetic limit order. Otherwise the queue position of the synthetic limit order advances.

3. Cancellations:

   - If a cancellation occurs in front of a synthetic limit order, then the queue position of the limit order advances.

This simulation approach avoids the aforementioned two issues of real limit orders. First, because synthetic limit orders are submitted uniformly over time, it avoids selection biases introduced by trading strategies. Second, because synthetic limit orders aren't canceled until the end of the day, the censoring of fill-time observations is vastly alleviated.

Figure 2.4 gives a graphic depiction of synthetic limit orders using historical Bank of America data over a particular trading day.

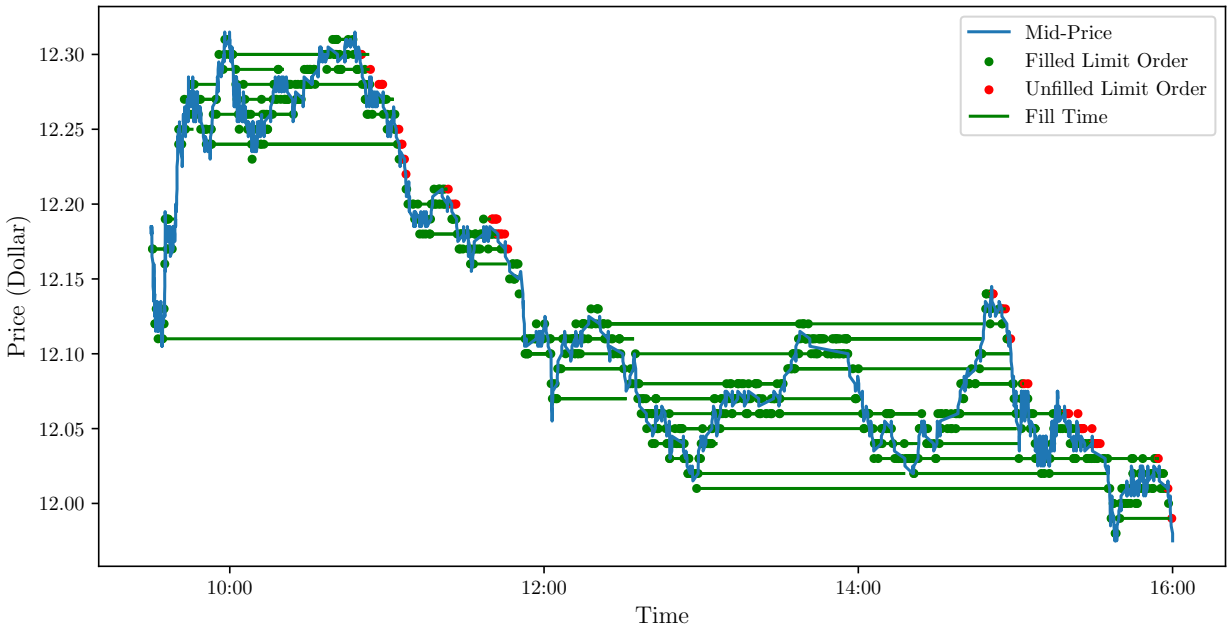


Figure 2.4: The blue line is the mid-price of BOA stock over the course of a day. Each synthetic limit order is represented by a dot at the time of submission. The dot is colored according to its execution outcome: If a limit order is filled by the end of the day, it is colored green; otherwise it is colored red. For any particular order that is filled, a horizontal line connects its submission time and its execution time — the length of the line represents the time-to-fill.

We record a limit order execution outcome Y as follows:

- A synthetic limit order is filled after time t: $Y = (\text{FILLED}, t)$.
- A synthetic limit order is not filled by time t and was cancelled automatically due to market close: $Y = (\text{UNFILLED}, t)$.

2.4.3 Maximum Likelihood Estimation

From the hazard rate setup, density and cumulative distribution can be explicitly derived. Therefore log-likelihood can be calculated for each limit order and maximum likelihood estimation can be used for training the RNN. The log-likelihood function can be expressed as follows:

- For an executed limit order: $\mathcal{L}(\lambda; (\text{FILLED}, t)) = \log(f_T(t; \lambda)) = -\Lambda(t) + \log\left(\lambda_{i^*(t)+1}\right)$.
- For an unexecuted limit order: $\mathcal{L}(\lambda; (\text{UNFILLED}, t)) = \log(F(t; \lambda)) = \log\left(1 - e^{\Lambda(t)}\right)$.

The hazard rates λ's are functions of RNN parameters θ, and therefore the log-likelihood function is ultimately a function of the RNN parameters θ. The RNN can be trained by maximizing the average log-likelihood across all synthetic limit orders

$$\max_{\theta} \ \mathbb{E}\left[\mathcal{L}\left(\theta; Y\right)\right].$$

2.5 Numerical Experiment Setup

This section outlines the details of the data used in the numerical experiments, including stock selection, limit order simulation, and train-test split procedure. Descriptive statistics are presented and some predictive patterns are discussed as well.

2.5.1 Stock Selection and Experiment Setup

The data we use is from the 502 trading days in the interval from October 1st 2012 to September 30th 2014. A set of large-tick U.S. stocks with high liquidity are selected for this study (see Table 2.4). For each trading day, 1000 synthetic limit orders are simulated at times chosen uniformly throughout the trading hours. For each synthetic limit order, the set of input features is collected and its execution outcome is recorded. These are used as inputs and outputs of the supervised learning algorithm.

For the purpose of fill time distribution estimation, we divide the time axis into 10 intervals (9 closed intervals and 1 half-open half-closed interval). The boundaries of these intervals are set to the deciles of the synthetic fill times.

We train and test a RNN model using 14 months of the data — first 12 months as training data, a subsequent month for validation, and the final month as testing data. The model is regularized by early stopping on the validation data set — the performance on the validation data set is monitored during the training process and the training is stopped once the performance stops improving. This procedure is repeated 10 times, with training, validation and testing data each advancing a month, until reaching the end of the data period (Sept. 30th 2014). Once the RNN models are trained, the

performances are computed on the testing data sets.

Ticker	Avg. Price($)	Vol.(%)	Volume($m)	One tick(%)	T-Size(s)	T-Size(%)
BAC	13.99	24.3	97.6	98.6	51548	0.73
GE	24.40	16.4	70.7	98.7	12047	0.42
VZ	47.94	16.3	85.4	99.1	2225	0.15
F	15.09	22.9	41.9	98.9	20248	0.73
INTC	24.97	20.9	137.1	98.4	13654	0.25
MSFT	34.56	22.1	236.6	97.7	10057	0.15
CSCO	22.52	21.6	131.8	99.1	17090	0.29
WFC	42.85	15.4	97.5	96.9	3433	0.15

Table 2.4: Descriptive statistics for the above 8 stocks over the two-year period. Average price and (annualized) volatility are calculated using daily closing price. Volume($m) is the average daily trading volume in the unit of a million dollars. One tick(%) is the percentage of time during the trading hours that the spread is one tick. Touch size(s) is the time average of the shares on the top price levels, averaged across the bid and ask. Touch size(%) is normalized using average daily volume, reflecting the percentage of daily liquidity that is available at the best prices.

2.5.2 Execution Statistics

The following statistics are average values across all 8 stocks over the two-year period. In the following discussion, we will focus on the two quantities below.

- Fill Probability: The probability that a limit order gets filled within a given time threshold h, in other words, $P(T < h)$.
- Conditional Fill Time: The expected fill time given an execution within the time threshold, mathematically expressed as $\mathbb{E}[T|T < h]$.

	Time Horizon		
Statistics	1 Min	5 Min	10 Min
Fill Probability	45%	76%	84%
Average Fill Time (sec)	22.5	70.0	103.4

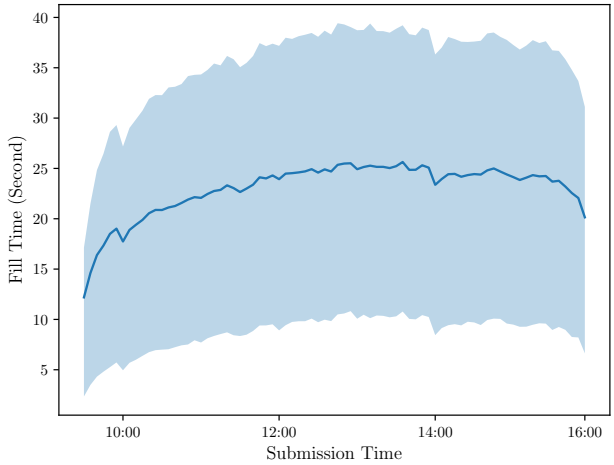
Table 2.5: Descriptive Statistics

2.5.3 Predictable Patterns

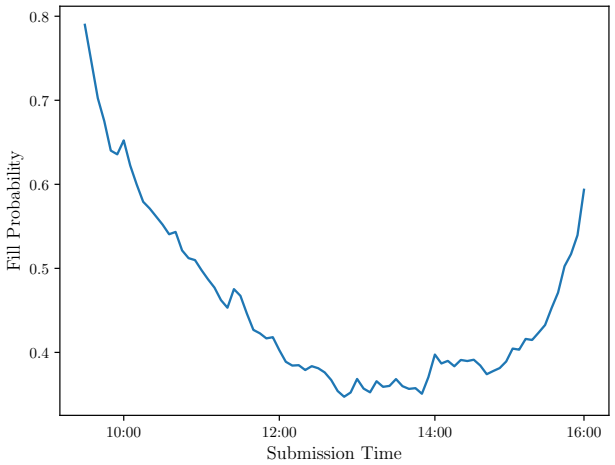
Even though limit order executions are inherently random events, there are some features that exhibit strong predictable patterns. These patterns motivate the selection of input features and the construction of benchmark models.

Time of Day:

Trading intensity exhibits intraday patterns. It is most intense around market open and market close, and slowest around noon. This general pattern has strong implication to limit order executions as well. Limit orders are executed faster and with higher fill probabilities around market open and close and are executed slower and with lower fill probabilities around noon. To demonstrate these patterns, trading hours are broken into 5-minute intervals, and for each interval, average conditional fill times and fill probabilities for all synthetic limit orders given a one-minute time horizon (h = 1 min), are plotted on Figure 2.5. This pattern persists for different time horizons.



(a) Fill times are shorter around market open and close, longer around noon.
The standard errors are less than 0.13 seconds.
The shaded area represents the 25% to 75% percentile.



(b) Fill probably is higher around market open and close, lower around noon.
The standard errors are less than 0.2%.

Figure 2.5: Time of Day Pattern (h = 1 min)

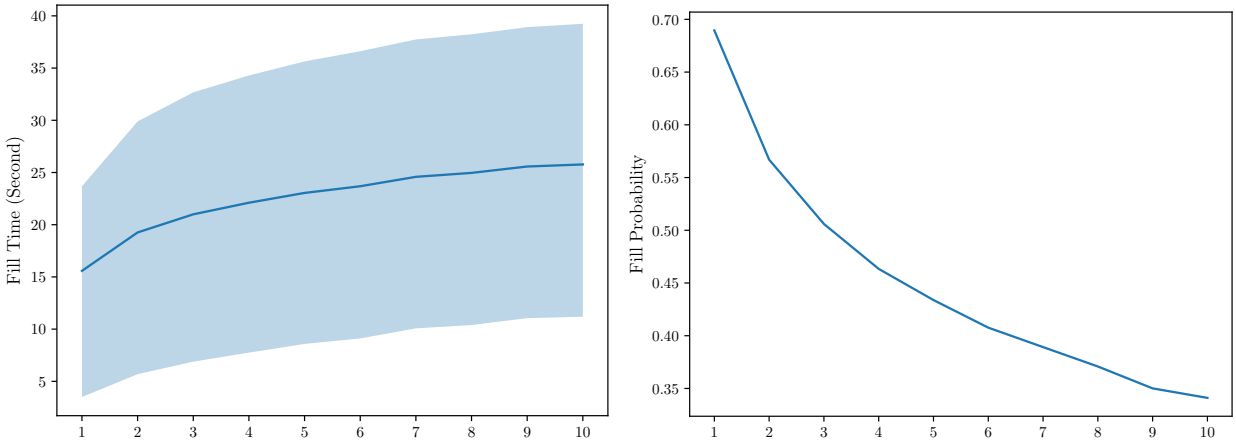
Queue Imbalance:

Queue imbalance (QI) is the percentage difference between queue lengths at the top price levels. Queue imbalance can be expressed mathematically as follows

$$\text{QI} = \frac{Q_{\text{near}} - Q_{\text{far}}}{Q_{\text{near}} + Q_{\text{far}}},$$

where Q_{near} and Q_{far} are the queue length at top price levels on the near side and the far side respectively. Queue imbalance reflects the instantaneous imbalance between the supply and the demand for the stock at the current price level. A negative queue imbalance signifies a stronger far side, and the price is more likely to move towards the near side to rebalance the supply and demand. This leads to a higher fill probability and a faster execution for orders submitted to the near side. Conversely, a positive queue imbalance signifies a stronger near side, and the price is more likely to move towards the far side. This leads to a lower fill percentage and a slower execution on average.

Figure 2.6 shows these patterns. The queue imbalance is recorded at the submission time for each synthetic limit order. These queue imbalance values are then divided into 10 deciles. The average fill times and fill probabilities of the synthetic limit orders submitted with queue imbalance within each deciles are computed and plotted in Figure 2.6.

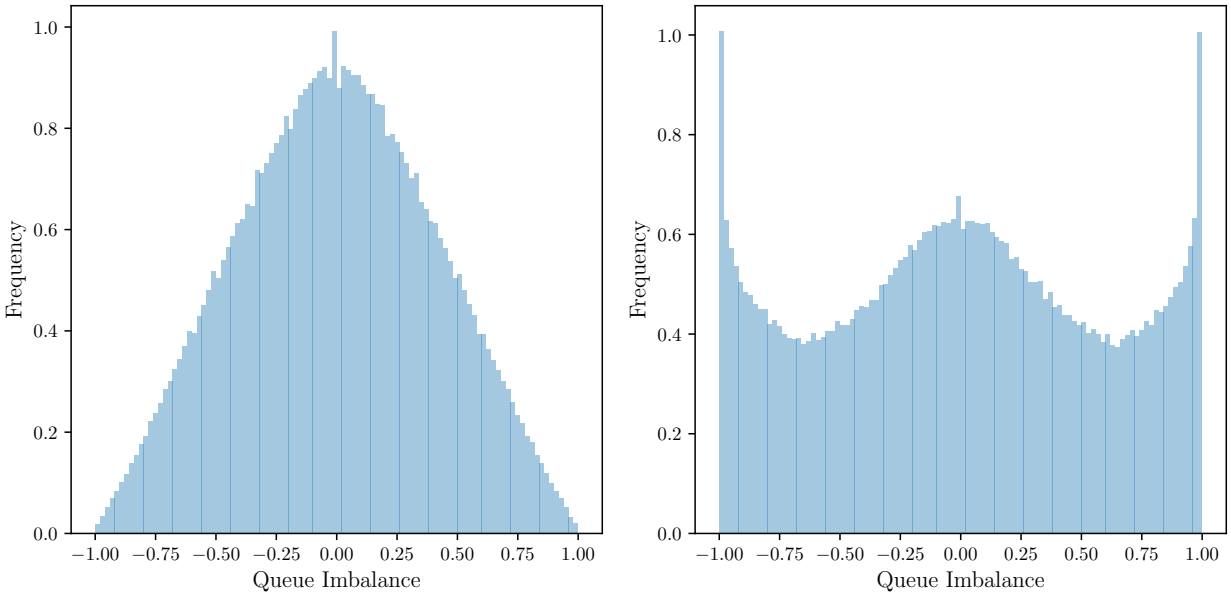


(a) Smaller QI lead to faster executions and large QI lead to slower executions.
The standard errors are less than 0.03 seconds.
The shaded area represents the 25% to 75% percentile.

(b) Smaller QI lead to higher fill probabilities and large QI lead to lower probabilities.
The standard errors are less than 0.01%.

Figure 2.6: Queue Imbalance Patterns (h = 1 min)

Another way to see the impact of queue imbalance is as follows. Trades don't occur uniformly over time. Rather, they occur more often when queue imbalance is at extreme values. Figure 2.7 illustrates this fact. The left histogram is of queue imbalance sampled at uniformly random time throughout trading days in our data period. The near/far side is also chosen randomly. Clearly, the histogram has a symmetric bell shape centered at 0. This implies that the supply and demand in the market are nearly balanced most of the time, and extreme imbalance occurs very rarely. The right histogram is of queue imbalance sampled only at moments of trades. The near/far side is also chosen randomly. The histogram is still symmetric and centered at 0, but it has much higher concentration at extreme values (close to -1 and 1).



(a) Queue imbalances sampled at uniformly random time.

(b) Queue imbalances sampled at trades times.

Figure 2.7: Queue Imbalance Histograms

2.6 Numerical Experiment Results

This section outlines the results of the numerical experiments. The performance of the RNN models are compared to benchmark models, and their applications in the trading problem of Section 2.2.2 is revisited.

2.6.1 Benchmark Models

We compare the performance of RNN against benchmark models using the same two metrics from Section 2.5.2, namely fill probability and conditional expected fill time. The following benchmark models are used for comparison purposes.

Linear/Logistic Regression:

Predicting whether an order will be filled is a binary classification problem and logistic regression is a natural linear benchmark. Predicting conditional expected fill time is a continuous value

prediction problem and linear regression is a natural benchmark. Only the input features collected at the time of submission are used in these two models.

Bucket Prediction:

Regressions only capture linear patterns in the data. To construct a non-parametric benchmark, we use bucketed empirical means as estimators. Based on the discussion in Section 2.5.3, we have chosen time of day and queue imbalance as features for bucketing.

Time of day is divided into 15-minute intervals and queue imbalance is divided into quintiles. Each bucket is the intersection of a time-of-day interval and a queue imbalance quintile. Within each bucket, simple empirical means of whether orders are filled and their fill times are used as the predictions.

Point Estimator:

For the problem of estimating fill time, the simplest estimation method wold be to make completely unconditional prediction with respect to market conditions. We call this the point estimator, it is computed by averaging fill times across all orders filled within a target horizon.

2.6.2 Fill Probability

To evaluate the accuracy of fill probability predictions using various models, the area under the curve (AUC) of a receiver operating characteristic (ROC) curve is used as a metric.

	Time Horizon		
AUC	1 min	5 min	10 min
Logistic Regression	0.62	0.60	0.60
Bucket Prediction	0.63	0.60	0.60
RNN	0.72	0.67	0.66

Table 2.6: Fill Probability Prediction Results

As we see in Table 2.6, RNN outperforms both benchmark models in all three time horizons. As the time horizon lengthens, the prediction accuracy of all three models decreases, suggesting

that the fill probability is more predictable within a shorter time horizons.

2.6.3 Conditional Expectation of Fill Time

To evaluate the accuracy of the predictions of conditional expectation of fill time, the root mean square error (RMSE) between actual fill times and estimated fill times is used as a metric.

	Time Horizon					
	1 min		5 min		10 min	
	RMSE	reduct.%	RMSE	reduct.%	RMSE	reduct.%
Point Estimator	16.6	-	70.2	-	123.1	-
Linear Regression	16.3	1.8%	68.9	1.9%	121.2	1.5%
Bucket Prediction	16.1	3.0%	68.3	2.7%	119.1	3.2%
RNN	15.1	9.0%	63.9	9.0%	112.8	8.4%

Table 2.7: Conditional Expected Fill-Time Prediction Results: The percentage reduction (reduct.%) is relative to the point estimator RMSE.

As we can see in Table 2.7, RNN outperforms all benchmark models in all three time horizons. Similar to fill probability prediction, as the time horizon lengthens, the prediction accuracy of all models decreases, suggesting that the fill-time is more predictable within a shorter time horizon.

2.6.4 Implementation Shortfall

In this section, we revisit the trading problem of Section 2.2.2 and demonstrate that a more accurate fill probability prediction leads to a better trading strategy in terms of reducing implementation shortfall.

As discussed in Section 2.2.2, a trader is seeking to buy a share of the stock with either a market order or a limit order. In order to minimize implementation shortfall, the optimal trading decision will be based on a threshold policy. More specifically, a limit order should be used if and only if

$$P(T \leq h) > \Theta,$$

where h is the target horizon, T is the random variable representing the fill time, $P(T \leq h)$ is the

fill probability, and Θ is the unknown threshold. The fill probability $P(T \leq h)$ can be estimated using various predictive models, and Θ will be optimized according to a method described shortly.

For a particular trade, the choice of limit order versus market order is a direct consequence of the fill probability prediction. This choice impacts the resulting implementation shortfall. Let $\hat{p}$ be the predicted fill probability within time horizon h. The implementation shortfall for the particular order can be expressed as a function of fill probability $\hat{p}$, horizon h, and the threshold Θ by

$$\text{IS} = \mathbb{1}\{\hat{p} \leq \Theta\}\text{IS}_{\text{mkt}} + \mathbb{1}\{\hat{p} > \Theta\}\text{IS}_{\text{limit}}, \tag{2.1}$$

where IS_{mkt} and IS_{limit} are the implementation shortfall of using a market order and a limit order respectively, given by

$$\text{IS}_{\text{mkt}} = p_A(0) - p_M(0); \quad \text{IS}_{\text{limit}} = \mathbb{1}\{T \leq h\}[p_B(0) - P_M(0)] + \mathbb{1}\{T > h\}\text{C}_{\text{clean-up}}. \tag{2.2}$$

At the limit order submission time, the mid-price is $p_M(0)$, the best ask price is $P_A(0)$, and the best bid price is $p_B(0)$. The clean-up trade occurs when the limit order is predicted to be executed within the target horizon, but doesn't actually execute. This condition can be expressed as $\mathbb{1}\{\hat{p} > \Theta\}\mathbb{1}\{T > h\} = 1$. Under this condition, the trader is required to fill the order with a market order at time h. The cost of this market order is the clean-up cost, given by

$$\text{C}_{\text{clean-up}} = p_A(h) - p_M(0), \tag{2.3}$$

where $p_A(h)$ is the best ask price at time h, which is the cost of a market order at the end of the target horizon.

To optimally set the threshold Θ, for each model, we choose the threshold that minimizes implementation shortfall empirically over the test data set.

Numerical Results:

Table 2.8 displays the average implementation shortfall per trade using various strategies. The market order strategy is when the trader submits market orders for all trades at the beginning of the time horizon. Because every trade yields half-spread as implementation shortfall, the length of the time horizon doesn't affect this strategy at all. The limit order strategy is when the trader submits limit orders for all traders at the beginning of the time horizon and places clean-up traders if the limit orders aren't executed by the end of the horizon. For longer time horizons, limit orders are executed with higher probability and this leads to a lower implementation shortfall. These two strategies do not require any predictive modeling.

We compare these to policies based on fill probabilities prediction using three different models — logistic regression, bucket prediction, and the RNN model. These three strategies follow the threshold policy with their respective fill probability estimates. Table 2.8 displays the implementation shortfall of these five trading strategies averaged over all synthetic limit orders.

	Time Horizon								
	1 min			5 min			10 min		
IS (ticks)	mean	s.e.	reduct.%	mean	s.e.	reduct.%	mean	s.e.	reduct.%
Market Order Strat.	0.508	-	-14%	0.508	-	-51%	0.508	-	-95%
Limit Order Strat.	0.443	0.02	-	0.337	0.04	-	0.260	0.07	-
Logistic Reg.	0.321	0.02	28%	0.198	0.04	41%	0.163	0.07	37%
Bucket Pred.	0.317	0.02	28%	0.194	0.04	42%	0.159	0.06	39%
RNN	0.279	0.02	37%	0.172	0.03	49%	0.148	0.06	43%

Table 2.8: Percentage reduction (reduct.%) is relative to the limit order strategy. The implementation shortfall displayed above is in ticks.

As can be seen in Table 2.8, using any predictive models significantly reduces implementation shortfall, with RNN yielding the best performance across all three time horizons. These reductions is partly due to a more accurate fill probability prediction, but also partly due to a smaller clean-up cost. Empirically, RNN consistently produces a lower clean-up cost compared to other models (see Table 2.9). This suggests that when RNN incorrectly predicts a limit order execution, the price doesn't move as far to the other direction as when the benchmark models make the same incorrect

prediction. For longer time horizons, because the price has more time to move away even further, the clean-up cost becomes much higher.

	Time Horizon					
	1 min		5 min		10 min	
Clean-up Cost (ticks)	mean	s.e.	mean	s.e.	mean	s.e.
Limit Order Strategy	1.22	0.01	3.01	0.02	4.29	0.05
Logistic Regression	1.17	0.01	2.95	0.02	4.25	0.05
Bucket Estimator	1.14	0.01	2.94	0.03	4.12	0.06
RNN	1.09	0.01	2.89	0.03	4.08	0.05

Table 2.9: Average Clean-up Cost (ticks) Conditioned on Clean-up

2.7 Conclusion

The choice between market orders and limit orders can be viewed a tradeoff between immediate executions and price premium. To make this choice intelligently, one must consider the uncertainty of limit order executions. In this study, we develop a data-driven approach to predict fill probabilities via estimating the distribution of limit order fill times.

In order to generate an unbiased data set of limit order fill times, we use historical NASDAQ ITCH dataset to simulate synthetic limit orders, track their positions, and record their fill times. To estimate the distribution of fill times, we construct a RNN to predict hazard rates on pre-determined intervals on the time axis. The RNN produces significant predictabilities, more accuracy than benchmark models. This prediction improvement has economic values as well. In a prototypical trading problem, when the trading strategy is implemented by RNN, it results the lowest implementation shortfall.

This study differs from many other studies in the following ways:

1. As far as we know, this is the first study to predict the distribution of limit order fill times.

2. By using a data-driven approach, we operate under minimal model assumptions.

3. We use a RNN to incorporate past order flow information for prediction.

The following are some remarks regarding limitations of this study:

1. Our current method only provides estimates for limit orders submitted to the best bid/ask price. Previous studies have found that the execution outcomes are sensitive to limit prices, and therefore it's inappropriate to use our current model to provide estimates for limit orders submitted at other price levels. A further study can be conducted by extending this chapter to multiple price levels. This would help evaluate a further tradeoff between limit prices and fill probabilities.

2. The synthetic limit orders are assumed to be infinitesimal and devoid of any market impact. This also implies that the synthetic limit orders can't be partially filled. However, previous studies have suggested that the size of the limit order doesn't impact the execution outcomes significantly.

Chapter 3: A Reinforcement Learning Approach to Optimal Execution

3.1 Introduction

Optimal execution is a classic problem in finance that aims to optimize trading while balancing various tradeoffs. When trading a large order of stock, one of the most common tradeoffs is between market impact and price uncertainty. More specifically, if a large order is submitted as a single execution, the market would typically move in the adverse direction, worsening the average execution price. This phenomenon is commonly referred to as the "market impact." In order to minimize the market impact, the trader has an incentive to divide the large order into smaller child orders and execute them gradually over time. However, this strategy inevitably prolongs the execution horizon, exposing the trader to a greater degree of price uncertainty. Optimal execution problems seek to obtain an optimal trading schedule while balancing a specific tradeoff such as this.

We will refer to the execution problem mentioned above as the parent order problem, where an important issue is to divide a large parent order into smaller child orders to mitigate market impact. In this paper, we focus on the optimal execution of the child orders, that is, after the parent order is divided, the problem of executing each one of the child orders. The child orders are quite different in nature compared to the parent order. The child orders are typically much smaller in size, and the prescribed execution horizons are typically much shorter. In practice, a parent order is typically completed within hours or days while a child orders are typically completed within seconds or minutes. Because any further dividing of an order can be viewed as another parent order problem, we will only consider the child order problem at the most atomic level. At this level, the child orders will not be further divided. In other words, each child order will be fulfilled in a single execution.

Because the market impact is negligible for a child order and the order must be fulfilled in a single execution, the most important aspect of the problem is the timing of the execution. More specifically, the trader seeks to execute the child order at an optimal time within the prescribed execution horizon. In this paper, we will develop data-driven approach based on price prediction to solve the execution timing problem.

The main contributions of this paper are as follows.

- **Execution Timing Problem.** We formulate the execution timing problem as an optimal stopping problem, where prediction of the future prices is an important ingredient.

- **Data-Driven Approach.** Unlike the majority of work in this area, we make no model assumptions on the price dynamics. Instead, we construct a novel neural network architecture that forecasts future price dynamics based on current market conditions. Using the neural network predictions, the trader can develop an execution policy.

  In order to implement the data-driven approach, we develop two specific methods, one based on supervised learning (SL), and the other based on reinforcement learning (RL). There are also different ways to train the neural network for these two methods. Specifically, empirical Monte Carlo (MC) and temporal difference (TD) learning can be applied and provide different variants of the SL and RL methods.

- **Backtested Numerical Experiments.** The data-driven approach developed in this paper is tested using historical market data, and is shown to generate significant cost saving. More specifically, the data-driven approach can recover a price gain of 20% of the half-spread of a stock for each execution in average, significantly reduce transaction costs.

  The RL method is also shown to be superior than the SL method when the maximal achievable performance is compared. There are a few other interesting insights that are revealed in the numerical experiments. Specifically, the choice of TD learning and MC update method presents various tradeoffs including convergence rates, data efficiency, and a tradeoff between bias and variance.

Through numerical experiments, we also demonstrate a certain universality among stocks in the limit order book market. Specifically, a model trained with experiences from trading one stock can generate non-trivial performance on a different stock.

3.1.1 Organization of the chapter

The rest of the chapter is organized as follows. Section 3.2 introduces the mechanics of limit order book markets and outlines the optimal stopping formulation. Section 3.3 introduces the supervised learning method and its induced execution policy. TD learning is also introduced in this section. Section 3.4 introduces the reinforcement learning method its induced execution policy. Section 3.5 outlines data source and the setup for the numerical experiments. Section 3.6 presents the numerical results and the various tradeoffs in training process introduced by TD learning. The aforementioned universality are also discussed in Section 3.6.

3.2 Limit Order Book and Optimal Stopping Formulation

3.2.1 Limit Order Book Mechanics

In modern electronic stock exchanges, limit order books are responsible for keeping track of resting limit orders at different price levels. Because investors' preferences and positions change over time, limit order books also need to be dynamics and changing over time. During trading hours, market orders and limit orders are constantly being submitted and traded. These events alter the amount of resting limit orders, consequently, the shape of the limit order book. There are other market events that alter the shape of the limit order book, such as order cancellation.

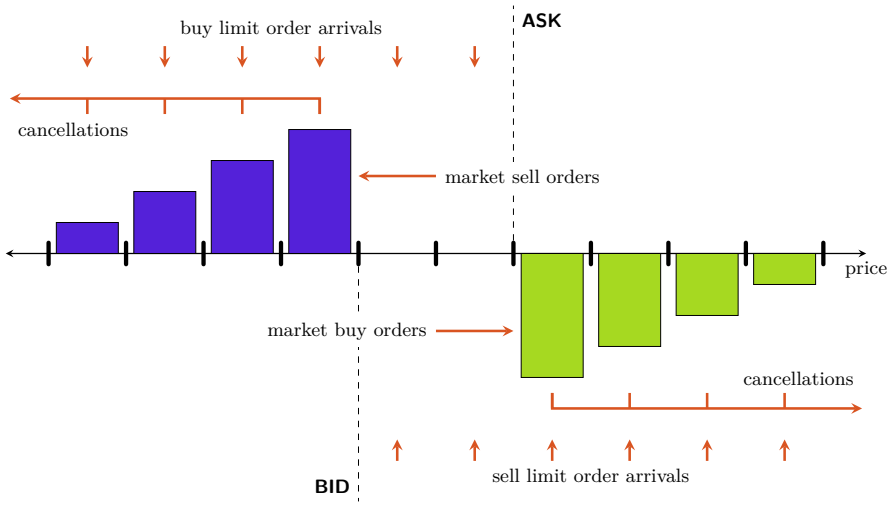


Figure 3.1: Limit orders are submitted at different price levels. The ask prices are higher than the bid prices. The difference between the lowest ask price and the highest bid price is the bid-ask spread. Mid-price is the average of the best ask price and the best bid price.

Limit order books are also paired with matching engines that match incoming market orders with resting limit orders to fulfill trades. The most common rule that the matching engine operates under is "price time priority." When a new market order has been submitted to buy, sell limit orders at the lowest ask price will be executed; when a new market order has been submitted to sell, buy limit orders at the highest bid price will be executed. For limit orders at the same price, the matching engine follows a time priority — whichever order was submitted first gets executed first.

3.2.2 Price Predictability

Some theoretical models in the classic optimal execution literature treat future prices as unpredictable. However, this doesn't always reconcile with market data. There is empirical evidence that stock prices can be predicted to a certain extent — Sirignano (2019) predicts the direction of price moves using a neural network and detects significant predictabilities.

Clearly, the ability to predict future prices would have major implications on stock executions. If a trader seeks to sell and predicts that the future price will move up, then the trader would have

an incentive to wait. On the other hand, if the trader predicts that the future price will drop, then the trader would have an incentive to sell immediately. In short, at least at a conceptual level, price predictability improves execution quality. This motivates us to construct a data-driven solution incorporating price predictability to optimal execution problems.

3.2.3 Optimal Stopping Formulation

Our framework will be that of a discrete-time sequential decision problem over a finite execution horizon T. The set of discrete time instances within the execution horizon is $\mathcal{T} \triangleq \{0, 1, ..., T\}$. For a particular stock, its relevant market conditions are represented by a discrete-time Markov chain with state $\{x_t\}_{t\in\mathcal{T}}$. We will assume that the transition kernel P is time-invariant. One of the state variables in the state that is of particular interest is the price of the stock, and we will denote this price process by $\{p_t\}_{t\in\mathcal{T}}$.

Consider the problem of selling one share of the stock, or equivalently, consider the order to be infinitesimal, that is, the order can't be further divided. This problem singles out the timing aspect of the execution and assumes that any action of the trader has no impact on the price process, the states, and the transitional kernel.

For a trader, the set of available actions at time t is $a_t \in \mathcal{A} = \{\text{CONTINUE, STOP}\}$. In other words, at any time instance, the trader can either hold the stock and continue to the next time instance, or sell the stock and stop. Because the trader is endowed with only 1 share of the stock, once the trader sells, no further action can be taken. In essence, this is an optimal stopping problem — the trader holds the stock and picks an optimal time to sell.

Let τ be a stopping time. Then, the sequence of states and actions before stopping is as follows

$$\{x_0, a_0, x_1, a_1, ..., x_\tau, a_\tau\}, \tag{3.1}$$

where a_τ = STOP by the definition of the stopping time. The trader's goal is to maximize the

expected total price difference between the execution price p_τ and the initial price, namely,

$$\max_{\tau} \quad \mathbb{E}[p_\tau - p_0]. \tag{3.2}$$

We will refer to this value as the total price gain and denote it by $\Delta P_\tau \triangleq p_\tau - p_0$. Maximizing the total price gain is equivalent to minimizing the implementation shortfall in this problem. Total price gain can be decomposed into the price gain between each time instance while the trader holds the stock. Let $\Delta p_t \triangleq p_t - p_{t-1}$. Then, the total price gain can be decomposed into per-period price difference,

$$\Delta P_\tau = \sum_{t=1}^{\tau} \Delta p_t. \tag{3.3}$$

From a sequential decision problem standpoint, this is not the only way to decompose the total price gain across time. One can also design a framework where the traders only receive a terminal reward when they stops. This decomposition approach benefits a learning agent by giving per-period rewards as immediate feedback.

Define a σ-algebra $\mathcal{F}_t \triangleq \sigma(x_0, a_0, ..., , x_{t-1}, a_{t-1}, x_t)$ for each time t, and a filtration $\mathcal{F} \triangleq \{\mathcal{F}_t\}_{t\in\mathcal{T}}$. Let random variable π_t be a choice of action that is $\mathcal{F}_t$-measurable and takes values in $\mathcal{A}$, and let a policy π be a sequence of such choices, i.e. $\pi = \{\pi_t\}_{t\in\mathcal{T}}$, and is $\mathcal{F}$-adapted. As constrained by the execution horizon, the last action must be STOP, i.e. $\pi_T \triangleq$ STOP.

Let Π be the set of all such policies, and an optimal policy π^* is given by

$$\pi^* \triangleq \operatorname*{argmax}_{\pi\in\Pi} \ \mathbb{E}_\pi \left[\sum_{t=1}^{\tau_\pi} \Delta p_t \right], \tag{3.4}$$

where τ_π is the first stopping time associated with policy π, and the expectation is taken assuming the policy π is used. Learning an optimal policy from data is the main machine-learning task that will be discussed in the next two sections.

3.3 Supervised Learning Approach

3.3.1 Price Trajectory Prediction

Future prices have important implications on execution policies. If a selling trader can predict that the future price is higher than the current price, the trader would wait and execute at a later time. If the future price is predicted to be lower than the current price, the selling agent should sell immediately. In this section, we will formulate this intuition more formally and construct a price prediction approach to optimal execution via supervised learning.

Given a fixed execution horizon T, it's insufficient to only predict the immediate price change in the short term — even if the price goes down, it could still move back up and rise even higher before the end of the execution horizon. Therefore, to obtain an optimal execution policy, it's imperative to obtain a price prediction for the entire execution horizon. This can be achieved by predicting price changes at each time instances. More specifically, define a price change trajectory as follows,

$$\text{Price Change Trajectory} \triangleq [\Delta p_1, \Delta p_2, ..., \Delta p_T]. \tag{3.5}$$

This gives rise to p_t through

$$p_t = p_0 + \sum_{i=1}^{t} \Delta p_i.$$

In the rest of the section, we will construct supervised learning models to predict the price change trajectory.

3.3.2 Supervised Learning Method

Define an observation episode as a vector of states and price changes, ordered in time as (3.6). This is the data observation upon which we will construct supervised learning models.

$$\text{Observation Episode} \triangleq \{x_0, \Delta p_1, x_1, \Delta p_2, ..., \Delta p_T, x_T\}. \tag{3.6}$$

In order to take an action at time 0, the trader needs a price change trajectory prediction at time 0 when the only observable state is x_0. Given any current state x, in order to predict the subsequent price change trajectory, we construct a neural network as follows. The neural network takes a single state x as input and outputs a vector of T elements, corresponding to the price change at each of the subsequent time instance. This neural network is represented as follows in (3.7).

$$\text{Neural Network:} \quad NN^{\phi}(x) = [u_1^{\phi}(x), u_2^{\phi}(x), ..., u_T^{\phi}(x)]. \tag{3.7}$$

The neural network parameter is denoted by ϕ, and the output neuron $u_i^{\phi}(x)$ corresponds to the price change Δp_i for all $1 \leq i \leq T$.

Given an observation episode such as (3.6), the mean squared error (MSE) between predicted price changes and actual price changes can be used as a loss function. That is

$$\mathcal{L}(\phi; x_0) = \frac{1}{T} \sum_{i=1}^{T} \left[\Delta p_i - u_i^{\phi}(x_0) \right]^2. \tag{3.8}$$

The neural network can be trained by minimizing (3.8) averaged over many observation episodes. After the neural network is trained, it can be applied to all states, giving a price change trajectory prediction at each time instance.

3.3.3 Execution Policy

Given a state x, the output of the neural network is a prediction of the subsequent price change trajectory. Summing up the price changes provides an estimate of the cumulative price change. Let $W_{t:T}(x)$ be the estimated maximum cumulative price change over all remaining time when the current time is t. For all $t \in \mathcal{T} \setminus \{T\}$, $W_{t:T}(x)$ can be expressed as

$$W_{t:T}(x) \triangleq \max_{1 \leq h \leq T-t} \sum_{i=1}^{h} u_i^{\phi}(x). \tag{3.9}$$

Notice, because the transitional kernel P is time-invariant, only the difference in indices $T - t$ matters in the value of $W_{t:T}(x)$, not the index t or T itself. At any time before T, if the future price trajectory rises higher than the current price, a selling trader would have an incentive to wait. Otherwise the trader should sell right away. This execution policy can be formally written as follows.

Supervised Learning Policy:

When the current time is t and the current state is x, define a choice of action π_t^{SL} as below.

$$\pi_t^{\text{SL}}(x) \triangleq \begin{cases} \text{CONTINUE} & \text{if} \quad W_{t:T}(x) > 0 \\ \text{STOP} & \text{otherwise.} \end{cases}$$

The execution policy induced by the SL method is the sequence of all such choices, given by

$$\pi^{\text{SL}}(\cdot) \triangleq \{\pi_t^{\text{SL}}(\cdot)\}_{t \in \mathcal{T}}. \tag{3.10}$$

Note that this policy is a Markovian policy in that this decision at time t is a function of the current state x_t. This policy is dependent on the neural network through the value of $W_{t:T}(\cdot)$. To apply this policy, a trader would apply each action function sequentially at each state until STOP is taken. More specifically, given a sequence of states, the stopping time is given by

$$\tau_{\pi^{\text{SL}}} \triangleq \min\{t \mid \pi_t^{\text{SL}}(x_t) = \text{STOP}\}. \tag{3.11}$$

The total price gain induced by this policy on the specific observation episode is $\Delta P_{\tau_\pi^{\text{SL}}} = p_{\tau_\pi^{\text{SL}}} - p_0$. Once the trader stops, no further action can be taken.

3.3.4 Temporal Difference Learning

The method discussed in Section 3.3.2 is a straightforward supervised learning method. However it has a few drawbacks. From a practical perspective, given any observation episode such as (3.6), only $\{x_0, \Delta p_1, \Delta p_2, ..., \Delta p_T\}$ is being used to train the neural network and $\{x_1, x_2, ..., x_T\}$ isn't

being utilized at all during the training process. This prompts us to turn to TD learning.

TD learning is one of the central ideas in RL (see Sutton and Barto (1998)) and it can be applied to supervised learning as well. Supervised learning uses empirical observations to train a prediction model, in this case, the price changes Δp_t. The price changes Δp_t are used as target values in the loss function (3.8). TD learning uses a different way to construct the loss function. In a neural network as in (3.7), offsetting outputs and state inputs correspondingly would result in the same prediction, at least in expectations. In other words, if the neural network is trained properly, the following is true for $0 \leq k \leq t-1$,

$$u_t^{\phi}(x_0) = \mathbb{E}\left[u_{t-k}^{\phi}(x_k)|x_0\right]. \tag{3.12}$$

In (3.12), the output $u_t^{\phi}(x_0)$ estimates the price change t time instances subsequent to the observation of the state x_0, namely, Δp_t. On the right side, the output $u_{t-k}^{\phi}(x_k)$ estimates of the price change $t-k$ time instances subsequent to the observation of the state x_k, and this also estimates Δp_t, coinciding with the left side.

This equivalence of shifting in time allows us to use current model estimates as target values to construct a loss function. This leads to a major advantage of TD learning, that is, TD learning updates a prediction model based in part on current model estimates, without needing an entire observation episode. To apply this more concretely in this case, the loss function for SL method can be reformulated as below for a specific observation episode.

$$\mathcal{L}(\phi; x_0) = \frac{1}{T}\left[\left(\Delta p_1 - u_1^{\phi}(x_0)\right)^2 + \sum_{i=2}^{T}\left(u_{i-1}^{\phi}(x_1) - u_i^{\phi}(x_0)\right)^2\right]. \tag{3.13}$$

Notice that $u_1^{\phi}(x_0)$ is still matched to the price change Δp_1. For $i \geq 2$, $u_i^{\phi}(x_0)$ is matched to the current model estimate with a time shift $u_{i-1}^{\phi}(x_1)$. In effect, instead of using the entire episode of price changes as the target values, TD uses $[\Delta p_1, u_1^{\phi}(x_1), u_2^{\phi}(x_1), ..., u_{T-1}^{\phi}(x_1)]$ as the target values, substituting all but the first element by current model estimates with x_1 as input. The loss function in (3.13) effectively reaffirms the equivalence in (3.12) using squared loss.

For every $1 \leq t \leq T$, (3.12) defines a martingale

$$\{u_t^{\phi}(x_0), u_{t-1}^{\phi}(x_1), ..., u_{t-k}^{\phi}(x_k), ..., u_1^{\phi}(x_{t-1})\}. \tag{3.14}$$

That is, conditioned on the current state, the expected value of future prediction k time instances ahead is equal to the current prediction of the same time instance. If the predictions exhibits predictable variability, in principle, the prediction model could be improved. TD learning with loss function in (3.13) can be viewed as a way of regularizing the prediction model to satisfy the martingale property in (3.12).

The data required to compute (3.13) is $(x_0, \Delta p_1, x_1)$, which is a subset of the observation episode. Any other consecutive 3-tuple of the form $(x_t, \Delta p_{t+1}, x_{t+1})$ can be used to compute (3.13) as well. Because TD learning requires only partial observations to compute the loss function, it allows us to update the neural network on the go.

Compared to the conventional SL method in Section 3.3.2, TD learning uses data more efficiently. Given the same amount of data, it updates the neural network many more times without using repeated data. In fact, given any observation episode such as (3.6), the loss function in (3.13) can be computed T times using all 3-tuples within the observation episode, updating the neural network T times. On the other hand, the conventional SL uses the loss function in (3.8) and can update the neural network only once. This advantage in data efficiency resolves the aforementioned data-wasting issue — TD utilizes all the state variables and price changes in an observation episode during training.

TD(m-step) Prediction:

We will refer to the updating method used in the conventional SL method outlined in Section 3.3.2 as the "empirical Monte Carlo (MC)" [1] update method. The MC update method trains a prediction model exclusively using samples from historical data observations. It turns out that

[1] In this paper, our Monte Carlo updates utilize empirical samples, and do not require a generative model as in typical Monte Carlo simulations.

there is a full spectrum of algorithms between TD and MC.

In (3.13), TD substitutes all but the first target value by current model estimates. This can be generalized to a family of TD methods by substituting fewer target values and keeping more observations. Specifically, we can construct a TD(m-step) method that uses m price changes and $T - m$ model estimates as target values. The loss function of TD(m-step) for a specific observation episode is

$$\mathcal{L}(\phi; x_0) = \frac{1}{T}\left[\sum_{i=1}^{m}\left(\Delta p_i - u_i^{\phi}(x_0)\right)^2 + \sum_{i=m+1}^{T}\left(u_{i-1}^{\phi}(x_m) - u_i^{\phi}(x_0)\right)^2\right]; \qquad m = 1, ..., T. \tag{3.15}$$

The data required to compute the above loss function is a $(m + 2)$-tuple, given by

$$(x_0, \Delta p_1, \Delta p_2, ..., \Delta p_m, x_m), \tag{3.16}$$

and this can also be generalized to any $(m + 2)$-tuple within the observation episode. TD(m-step) updates the neural network $T + 1 - m$ times using one observation episode.

Notice, when $m = T$, (3.15) becomes the same as (3.8). In other words, TD(T-step) is the same as Monte Carlo. When $m = 1$, TD(1-step) has the loss function in (3.13), representing the highest level of TD. The TD step m is a hyper-parameter that controls the degree of TD when training the neural network. We will discuss the effect of the TD step m in greater detail in Section 3.6.2.

Double Q-Learning:

Neural networks are typically trained using stochastic gradient descent (SGD). However, (3.13) and (3.15) aren't suitable for SGD. When the parameter ϕ changes, both the prediction model and the target values change. To get around this issue, the idea of double Q-learning was introduced by van Hasselt, Guez, and Silver 2016. Instead of a single neural network, we maintain two neural networks, one for training purposes and the other for using as target values. These two neural networks need to have identical architectures and we denote their parameters by ϕ and ϕ',

respectively,

$$\text{Train-Net:} \quad NN^{\phi}(x) = [u_1^{\phi}(x), u_2^{\phi}(x), ..., u_T^{\phi}(x)]$$
$$\text{Target-Net:} \quad NN^{\phi'}(x) = [u_1^{\phi'}(x), u_2^{\phi'}(x), ..., u_T^{\phi'}(x)].$$

The train-net's parameter ϕ is the model that SGD changes during each iteration and the target-net is used exclusively for producing target values. The loss function can be written as

$$\mathcal{L}(\phi; x_0) = \frac{1}{T}\left[\sum_{i=1}^{m}\left(\Delta p_i - u_i^{\phi}(x_0)\right)^2 + \sum_{i=m+1}^{T}\left(u_{i-1}^{\phi'}(x_m) - u_i^{\phi}(x_0)\right)^2\right]; \qquad m = 1, ..., T. \tag{3.17}$$

The target-net also needs to be updated during the training so that it always provides accurate target values. Therefore, the train-net needs to be copied to the target-net periodically throughout the training procedure. The entire algorithm is outlined below in Section 3.3.5.

3.3.5 Algorithm

To summarize, the complete algorithm using supervised learning with TD(m-step) is displayed below. This algorithm will be referred to as the SL-TD(m-step) algorithm in the rest of this chapter.

Algorithm 1: SL-TD(m-step)

Initialize ϕ and ϕ' randomly and identically;
while not converged **do**
1. From a random episode, select a random starting time t, sample a sub-episode $(x_t, \Delta p_{t+1}, ..., \Delta p_{t+m}, x_{t+m})$ for $0 \le t \le T - m$;
2. Repeat step 1 to collect a mini-batch of sub-episodes;
3. Compute the average loss value over the mini-batch using (3.17);
4. Take a gradient step on ϕ to minimize the average loss value;
5. Copy target-net with train-net ($\phi' \leftarrow \phi$) periodically;

end

To monitor the training progression, in-sample and out-of-sample MSE can be computed and monitored. Each iteration of neural network parameter ϕ induces a corresponding execution policy. Applying this execution policy to observation episodes either in sample or out of sample gives the

price gains on these episodes. This measure of average price gains on observation episodes can also be used to monitor the training progression.

3.3.6 Insufficiency

We will use a hypothetical example to illustrate the insufficiency of the SL method outlined above. Let there be two possible future scenarios A and B for the price of a particular stock. Under these two scenarios, price change trajectories over the next two time instances are

$$\Delta P_A = [\Delta p_1^A, \Delta p_2^A] = [+1, -4]; \quad \Delta P_B = [\Delta p_1^B, \Delta p_2^B] = [-2, +3].$$

Assume that these two scenarios occur with equal probability given all current information, namely,

$$P(A|x_0) = P(B|x_0) = 0.5.$$

Given this information, the ex-post optimal execution would be to sell at $t = 1$ under scenario A and sell at $t = 2$ under scenario B. This execution plan would yield an execution price of $+1$ under either scenario.

Now consider applying the SL method when only the state x_0 is observable. The neural network is trained using MSE and it's well known that the mean minimizes MSE. In other words, the optimal prediction would be

$$NN^{\phi}(x_0) = [u_1^*(x_0), u_2^*(x_0)] = P(A|x_0) \cdot \Delta P_A + P(B|x_0) \cdot \Delta P_B = [-0.5, -0.5].$$

This prediction indicates that future price changes will always be negative and therefore the trader should sell at $t = 0$ and induce an execution price of 0.

It's not a surprise that the ex-ante execution is inferior compared to the ex-post execution. However, this example also reveals a rather unsatisfactory aspect of the SL method — even with "optimal prediction," the SL method fails to capture the optimal execution. The trader stops too

early and misses out on future opportunities.

3.4 Reinforcement Learning Approach

The SL method outlined above predicts the future price change trajectory for each state using neural networks. The predicted price change trajectory induces an execution policy, which can be applied sequentially to solve the optimal execution problem. However, the SL method doesn't lead to an optimal policy, which prompts us to turn to reinforcement learning (RL).

The insufficiency of the SL method discussed in Section 3.3.6 is mainly caused by the way SL averages predictions. SL produces an average prediction by simply averaging the price change trajectories under all possible scenarios, disregarding the fact that a trader might take different sequence of actions under each scenario. If the trader predicts a future price downturn, then the trader would stop and sell early. However, this price downturn, even though it can be predicted and avoided by the trader, is still accounted for in the SL prediction. In the example outlined in Section 3.3.6, Δp_2^A is one such price downturn. Including price downturns that can be predicted and avoided in the model predictions lead to a suboptimal policy.

RL averages trajectories from future scenarios differently. Instead of averaging the trajectories directly, RL allows the trader to take different sequence of actions under each scenario, and averages the resulting rewards. This way, if a price downturn can be predicted and avoided by the trader, it won't be accounted for in the RL prediction. This leads to an improved execution policy compared to the SL method.

However, RL adds more complexity to the algorithms, especially during the training process. SL predicts price change trajectories, which are exogenous to the trader's policy. During training, as SL prediction becomes more accurate, the induced policy improves. On the other hand, because RL predicts future rewards, which are dependent on the execution policy, the target values of the prediction are no longer exogenous. While the RL model is being trained, the induced policy changes accordingly, which in turns also affects the future rewards. We will discuss how this difference complicates the training procedure in the rest of this section.

The procedure of applying RL to the sequential decision problem isn't all that different compared to the SL method. RL also uses neural networks to evaluate the "value" or "quality" of each state, which leads to an execution policy that can be applied sequentially. The main difference is that instead of predicting price change trajectories, RL predicts what's called continuation value.

3.4.1 Continuation Value

Continuation value is defined as the expected maximum reward over all remaining time instances when the immediate action is CONTINUE. Specifically, we write $C_{t:T}(x)$ to denote the continuation value when the current time is t and the current state is x. For all $t \in \mathcal{T} \setminus \{T\}$, this is defined as

$$C_{t:T}(x) \triangleq \sup_{\pi \in \Pi^0_{t:T}} \mathbb{E}_\pi \left[\sum_{i=t}^{\tau_\pi} \Delta p_i \middle| x_t = x \right]. \tag{3.18}$$

The set $\Pi^0_{t:T}$ contains all policies starting from time t that don't immediately stop at time t, i.e.

$$\Pi^0_{t:T} = \{(\pi_t, \pi_{t+1}, ..., \pi_T) \,\big|\, \pi_t = \text{CONTINUE}\}. \tag{3.19}$$

The stopping time τ_π is the stopping time associated with policy π and the expectation is taken assuming the policy π is used. Notice that for any fixed x, the value of $C_{t:T}(x)$ depends on the pair (t, T) only through $T - t$. By convention, $C_{t:t}(x) \triangleq 0$ for all states x and times t.

Optimal Policy:

Because the future price gain of STOP is always 0, the definition of the continuation value leads to a very simple execution policy — the trader should continue if and only if the continuation value is strictly larger than 0. At time t, if the current state is x, define an action function as

$$\pi^{\text{RL}}_t(x) \triangleq \begin{cases} \text{CONTINUE} & \text{if} \quad C_{t:T}(x) > 0 \\ \text{STOP} & \text{otherwise.} \end{cases}$$

The execution policy induced by the RL method is the sequence of such action functions defined at all time instances,

$$\pi^{\mathrm{RL}}(\cdot) = \{\pi_t^{\mathrm{RL}}(\cdot)\}_{t\in\mathcal{T}}. \tag{3.20}$$

When applying this policy sequentially to a sequence of states, the associated stopping time and the total price gain is given by

$$\tau_{\pi^{\mathrm{RL}}} \triangleq \min\{t \,\big|\, \pi_t^{\mathrm{RL}}(x_t) = \mathrm{STOP}\}; \quad \Delta P_{\tau_\pi^{\mathrm{RL}}} = p_{\tau_\pi^{\mathrm{RL}}} - p_0. \tag{3.21}$$

Bellman Equation:

The above definition of the continuation value leads to the following properties.

1. If the current time is $T-1$, there is only one time instance left and the continuation value is the expectation of the next price change,

$$C_{T-1:T}(x) = \mathbb{E}[\Delta p_T | x_{T-1} = x]. \tag{3.22}$$

2. If there is more than one time instance left, the continuation value is the sum of the next price change and the maximum rewards achievable over all remaining time. This leads to a Bellman equation given by

$$C_{t:T}(x) = \mathbb{E}\left[\Delta p_{t+1} + \max\{0, C_{t+1:T}(x_{t+1})\} | x_t = x\right]. \tag{3.23}$$

If the trader follows the optimal policy starting at time $t+1$, the total reward accumulated after $t+1$ is precisely $\max\{0, C_{t+1}(x_{t+1})\}$. This is how (3.23) implicitly incorporated the execution policy.

Monotonicity:

At any time, the continuation value can be decomposed into a sum of increments of continuation values of stopping problems of increasing horizon. Because increasing time horizon allows more flexibility in trader's actions and can only increase the value of a stopping problem, these increments are non-negative. In other words, continuation values have a certain monotonicity.

At any time t, for any $i \geq 1$, define the continuation value increment as

$$\delta_i(x) \triangleq C_{t:t+i}(x) - C_{t:t+i-1}(x). \tag{3.24}$$

This is the difference in continuation values when the time horizon is $i-1$ time steps away and one time step is added. Then, for $i > 1$,

$$\delta_i(x) \geq 0. \tag{3.25}$$

When $i = 1$, the continuation value increment $\delta_i(x) = C_{t:t+1}(x) - C_{t:t}(x) = \mathbb{E}[\Delta p_{t+1} | x_t = x]$ can be negative. Summing up these increments recovers the continuation value, namely,

$$C_{t:T}(x) = \sum_{i=1}^{T-t} \delta_i(x). \tag{3.26}$$

3.4.2 Learning Task

Unlike the price trajectory, the continuation value isn't directly observable from the data. Furthermore, the continuation value is dependent on the induced policy and the induced policy evolves as the continuation value changes. For these reasons, learning the continuation value is not a conventional supervised learning task.

In order for the induced policy to apply to each time instance in the sequential decision problem, the continuation value $C_{t:T}(x)$ needs to be estimated for each $t = 0, \ldots, T-1$. We design a neural network to learn the continuation value from data. Because the parameter t is discrete and have

a fixed range, we incorporate this parameter directly into the neural network architecture. More specifically, the neural network takes a state x as an input and outputs a vector of T elements, representing each of the continuation value increment δ_i. The neural network can be represented as

$$\text{Neural Network:} \quad NN^{\phi}(x) = [u_1^{\phi}(x), u_2^{\phi}(x), ..., u_T^{\phi}(x)]. \tag{3.27}$$

This neural network contains T neurons on the output layer, and each neuron $u_i^{\phi}(x)$ is meant to approximate $\delta_i(x)$. As a result of this construction, the estimated continuation value is the summation of the neuron outputs, given by

$$\hat{C}_{t:T}^{\phi}(x) = \sum_{i=1}^{T-t} u_i^{\phi}(x). \tag{3.28}$$

There are two benefits of this neural network construction. One is that by incorporating time t as part of the architecture, it captures the commonality among continuation values for the entire time horizon. Secondly, due to the monotonicity of the continuation values, the increments $\delta_i(x)$ should always be non-negative for $i > 1$. This implies that the true value of $u_i^{\phi}(x)$ is non-negative for $i > 1$. Using this architecture, we can also easily enforce this positivity on the output layer by applying the softplus activation function. This way the neural network output is consistent with the monotonicity property.

In order to train the neural network, we need to construct target values from observation episodes as in (3.6). We can compute the "empirical continuation value" at time t when the current state is x_t, given by

$$\tilde{C}_{t:T}(x_t) = \Delta p_{t+1} + \sum_{i=2}^{T-t} \Delta p_{t+i} \cdot \Pi_{j=1}^{i-1} \mathbb{1}\{\hat{C}_{t+j:T}^{\phi}(x_{t+j}) > 0\}, \tag{3.29}$$

where $\hat{C}^{\phi}$ is the continuation value estimated from the current neural network using (3.28). The right side of (3.29) includes the immediate price change Δp_{t+1} and other price changes conditionally. The price change Δp_{t+i} is only accounted for if the trader reaches time $t + i$. Because the trader follows the execution policy induced by the current model, this condition is expressed as

$\Pi_{j=1}^{i-1} \mathbb{1}\{\hat{C}^{\phi}_{t+j:T}(x_{t+j}) > 0\}$.

The difference in the empirical continuation values is the empirical increments, given by

$$\tilde{\delta}_i(x) = \tilde{C}_{t:t+i}(x) - \tilde{C}_{t:t+i-1}(x). \tag{3.30}$$

This is the target value for $u_i^{\phi}(x)$. Now that the target values for the neural network outputs are in place, we can compute the mean squared error (MSE) loss function according to (3.31) and apply SGD to train the network parameters.

$$\mathcal{L}(\phi; x) = \frac{1}{T} \sum_{i=1}^{T} \left[u_i^{\phi}(x) - \tilde{\delta}_i(x)) \right]^2. \tag{3.31}$$

3.4.3 Temporal Difference Learning

The empirical continuation values can be obtained through TD learning as well. Instead of using empirical observations of price changes as in (3.29), the current model estimates of continuation values can be used to compute the empirical continuation values.

As described in Section 3.3.4, two neural networks are used, one for training and one for evaluating target values, given as follow,

$$\begin{aligned} &\text{Train-Net:} \quad NN^{\phi}(x) = [u_1^{\phi}(x), u_2^{\phi}(x), ..., u_T^{\phi}(x)] \\ &\text{Target-Net:} \quad NN^{\phi'}(x) = [u_1^{\phi'}(x), u_2^{\phi'}(x), ..., u_T^{\phi'}(x)]. \end{aligned}$$

According to TD(1-step), the empirical continuation value $\tilde{C}_{t:T}(x_t)$ is the sum of the immediate price change Δp_{t+1} and the estimated continuation value at evaluated at state x_{t+1}, conditional on the trader reaching time $t+1$. This is given by

$$\tilde{C}_{t:T}(x_t) = \Delta p_{t+1} + \hat{C}^{\phi'}_{t+1:T}(x_{t+1}) \cdot \mathbb{1}\{\hat{C}^{\phi}_{t+1:T}(x_{t+1}) > 0\}; \quad \forall t \leq T-1. \tag{3.32}$$

Notice that on the right side of (3.32), the policy is induced using the train-net and the continua-

tion value accumulated is evaluated from the target-net. This idea is commonly referred to as "Double Q-Learning," which is introduced by van Hasselt, Guez, and Silver (2016) to avoid the overestimation of action values in general DQN algorithms. The data used in (3.32) is $\{x_t, \Delta p_{t+1}, x_{t+1}\}$, which naturally extends to any 3-tuple of the same form.

The TD(m-step) can be applied as well, which expresses the empirical continuation value as

$$\tilde{C}_{t:T}(x_t) = \Delta p_{t+1} + \sum_{i=2}^{m} \Delta p_{t+i} \cdot \Pi_{j=1}^{i-1} \mathbb{1}\{\hat{C}^{\phi}_{t+j:T}(x_{t+j}) > 0\} + \hat{C}^{\phi'}_{t+m:T}(x_{t+m}) \Pi_{j=1}^{m} \mathbb{1}\{\hat{C}^{\phi}_{t+j:T}(x_{t+j}) > 0\}; \quad \forall t \leq T-m. \tag{3.33}$$

The data used in (3.33) is a $(2m+1)-$tuple

$$\{x_t, \Delta p_{t+1}, x_{t+1}, ..., \Delta p_{t+m}, x_{t+m}\}, \tag{3.34}$$

which can be generalized to any $(2m+1)-$tuple of the same form.

For TD(m-step), if the current time t is larger than $T-m$, then the current model estimates are no longer used as an additive terms in the computation of the target value. The target value of the continuation value is simply given by (3.29).

These TD methods computes the empirical continuation values, which leads to empirical increments. These increments help train the network networks as target values through the loss function in (3.31).

3.4.4 Algorithm

To summarize, the complete algorithm using reinforcement learning with TD(m-step) is displayed below. This algorithm will be referred to as the RL-TD(m-step) algorithm in the rest of this paper.

Algorithm 2: RL-TD(m-step)

Initialize ϕ and ϕ' randomly and identically;

while not converged **do**

1. From a random episode, select a random starting time t, and sample a sub-episode $(x_t, \Delta p_{t+1}, ..., \Delta p_{t+m}, x_{t+m})$ for $0 \leq t \leq T - m$;

2. Repeat step 1 to collect a mini-batch of sub-episodes;

3. Compute empirical continuation value increments and the average loss values using (3.31);

3. Take a gradient step on ϕ to minimize the average loss value;

3. Copy target-net with train-net ($\phi' \leftarrow \phi$) periodically;

end

When compared to the SL method, one critical difference in the RL method is that the target values for training the neural network is dependent on the induced policy. Therefore, the target value also changes during the training process. As a result, it's more difficult and perhaps less meaningful to monitor the MSE loss value during training. In order to monitor the training progress, the induced policy can be applied to observation episodes either in sample or out of sample to produce price gains.

3.4.5 Discussion

The optimal stopping problem is challenging because the future prices are stochastic. A simplification would be to make the problem deterministic. The simplest deterministic model consistent with the stochastic dynamics is to replace random quantities with their expectations. In particular, at each time t, we replace the stochastic future price trajectory with its mean. In this context, the continuation value becomes

$$C_{t:T}^{MPC}(x) = \max_{1 \leq h \leq T-t} \mathbb{E}\left[\sum_{i=1}^{h} \Delta p_{t+i} \middle| x_t\right]. \tag{3.35}$$

This motivates the SL method and the execution policy based upon (3.9). Notice that $C_{t:T}^{MPC}(x)$ is an underestimate of the true continuation value defined in (3.18), because it only optimizes over deterministic stopping times, while the true continuation value allows random stopping times. In other words,

$$C_{t:T}^{MPC}(x) \leq C_{t:T}(x).$$

The general idea of resolving a new, deterministic control problem at each instance of time goes under the rubric of "Model Predictive Control (MPC)." See Akesson and Toivonen (2006), for example, an application of neural networks in MPC.

Another simplification of the stochastic dynamics of the future price would be to use information relaxation, i.e., giving the trader the perfect knowledge of future price changes. There is a literature of information relaxation applied to stopping problems, which was pioneered by Rogers (2003) and Haugh and Kogan (2004). In our context, one information relaxation would be to reveal all future prices to the decision maker. This would make the problem deterministic, and result in

$$C_{t:T}^{IR}(x) = \mathbb{E}\left[\max_{1\leq h\leq T-t} \sum_{i=1}^{h} \Delta p_{t+i} \middle| x_t\right] \tag{3.36}$$

as the value of continuation. This is clearly an overestimation of the true continuation because it optimizes with access to future information, while the true continuation value expressed in (3.18) only optimizes with access to information that is currently available.

$$C_{t:T}(x) \leq C_{t:T}^{IR}(x).$$

A regression approach can formulated to estimate this maximum value, and this is in the spirit of Desai, Farias, and Moallemi (2012). We don't pursue this idea in this paper.

We can compare our method with the earlier work on optimal stopping problems in Longstaff and Schwartz (2001) and Tsitsiklis and Van Roy (2001). In these regression-based methods, by using a backward induction process that increases horizon one step at a time, a separate regression

model is fitted for each time horizon. In our method, we fit a nonlinear neural network that predicts the continuation value for all time horizons. By using neural networks, the model is able to capture common features across time horizons. Additionally, in the RL method, due to the monotonicity, we know the incremental values δ_i are positive. We apply softplus activation function on the output layer to enforce the positivity of the neural network outputs. This way, the estimated continuation values produced by the neural network also possess the desired monotonicity.

The idea of TD learning also manifest in the regression-based methods as well. Longstaff and Schwartz (2001) approximates the continuation value when the horizon is t using continuation values when the horizon is $t - 1$. This is similar to the idea of RL-TD(1-step). Tsitsiklis and Van Roy (2001) approximates the continuation value when the horizon is t using future rewards under policies determined when the horizon was $t - 1$. This is similar to the spirit of Monte Carlo update or RL-TD(T-step).

3.5 Numerical Experiment: Setup

The following two sections discuss the numerical experiments that test the SL and RL method discussed above. This section will outline the setup of these experiments including the data source, features, and neural network architectures.

3.5.1 NASDAQ Data Source

The NASDAQ ITCH dataset provides level III market data from the NASDAQ stock exchange (see NASDAQ 2010). This dataset contains event messages for every event that has transpired at the exchange. Common market events include "add order," "order executed," and "order cancel." These market events occur throughout the trading hours and constantly change the limit order book (LOB). An example of an "add order" event message is shown below in Table 3.1.

From these event messages, a limit order book can be constructed to display the prices and numbers of resting shares (depth) at each price level. This system is dynamic and it changes every time a new event occurs in the market.

time	ticker	side	shares	price	event
9:30:00.4704337	BAC	B	2000	12.02	"A"

Table 3.1: The event reads: A bid limit order of 2000 shares of BAC stock is added to the LOB at price level $12.02 at 9:30:00.4704337.

time	b.prc 5	b.prc 4	b.prc 3	b.prc 2	b.prc 1	a.prc 1	a.prc 2	a.prc 3	a.prc 4	a.prc 5
9:30:00.4704337	12.01	12.02	12.03	12.04	12.05	12.06	12.07	12.08	12.09	12.10
9:30:00.8582938	12.01	12.02	12.03	12.04	12.05	12.06	12.07	12.08	12.09	12.10

Table 3.2: The above table is a snapshot of the LOB displaying the prices of the top 5 price levels on both sides of the market before and after the event from Table 3.1. The event from Table 3.1 doesn't change the prices at each level.

The limit order book reflects the market condition at any given moment and this provides the environment of the optimal execution problem.

3.5.2 Experiment Setup

The dataset we use is over the entire year of 2013, which contains 252 trading days. A set of 50 high-liquidity stocks are selected for this study. The summary statistics for these 50 stocks can be seen in the Appendix (see Table A.1).

For each stock, 100 observation episodes are sampled within each trading day, with the starting time uniformly sampled between 10am and 3:30pm New York time. Each episode consists of 60 one-second intervals. In other words, the time horizon is one minute and $T = 60$.

The dataset of observation episodes is then randomized into three categories, a training dataset (60%), a validation dataset (20%) and a testing dataset (20%). The randomization occurs at the level of a trading day. In other words, no two episodes sampled from the same day would belong

time	b.prc 5	b.prc 4	b.prc 3	b.prc 2	b.prc 1	a.prc 1	a.prc 2	a.prc 3	a.prc 4	a.prc 5
9:30:00.4704337	10000	43700	13100	12100	7500	5200	15300	15900	17000	22200
9:30:00.8582938	10000	41700	13100	12100	7500	5200	15300	15900	17000	22200

Table 3.3: The above table is a snapshot of the LOB displaying the number of shares on the top 5 price levels on both sides of the market before and after the event from Table 3.1. The event from Table 3.1 reduces 2000 shares at price $12.02.

to two different categories. This is to avoid using future episodes to predict past episodes within the same day, as it violates causality.

The randomization setup allows the possibility of using future days' episodes to predict past days' price trajectories. However, because the execution horizon is as short as a minute and the features selected mostly capture market microstructure, we deem the predictabilities between different days as negligible.

We consider two regimes under which the models can be trained and tested. One is the "stock-specific" regime where a model is trained on a stock and tested on the same stock. The other is the "universal" regime where all the data of 50 stocks is aggregated before training and testing. This regime presumes that there is certain universality in terms of the price formation process across stocks. Specifically, the experiences learned from trading one stock can be generalized to another stock.

3.5.3 State Variables and Rewards

State Variables:

In a limit order book market, past events and current market conditions have predictive power for the immediate future. In order to capture this predictability, we have extracted a set of features from the order book to represent state variables. The complete set of features can be found in the Appendix (see Table A.2).

To better capture the temporal pattern in the market events, this set of features is collected not only at the current time, but also at each second for the past 9 seconds. The collection of these 10 sets of features collectively represent the market condition and are used as the state variable. More specifically, let s_t be the set of features collected at time t. Then the state variable $x_t = (s_{t-9}, s_{t-8}, ..., s_t)$ is a time series of these features, prior to time t.

Normalized Price Changes/Rewards:

We selected a diverse range of stocks with an average spread ranging from 1 tick to more than 54 ticks. The magnitudes of the price changes of these stocks also varied widely. As a result, it's inappropriate to use price changes directly as rewards when comparing different stocks. Instead, we normalized the price changes by the average half-spread, and use these quantities as rewards. In effect, the price gains are computed in units of percentage of the half-spread. If the price gain is exactly the half-spread, then the trade is executed at the mid-price. Thus, if the normalized price gain achieves 100%, then the trader is effectively trading frictionlessly.

Recurrent Neural Network (RNN):

RNN is specifically designed to process time series of inputs (see Figure 3.2). Sets of feature are ordered temporally and RNN units connect them them horizontally. The output layer is of dimension 60, matching the time horizon T. For the RL method, the monotonicity of the continuation value implies that the output neurons are non-negative except the $u_1^{\phi}(x)$. To enforce this positivity, the softplus activation function is applied to the output layer in the RL settings.

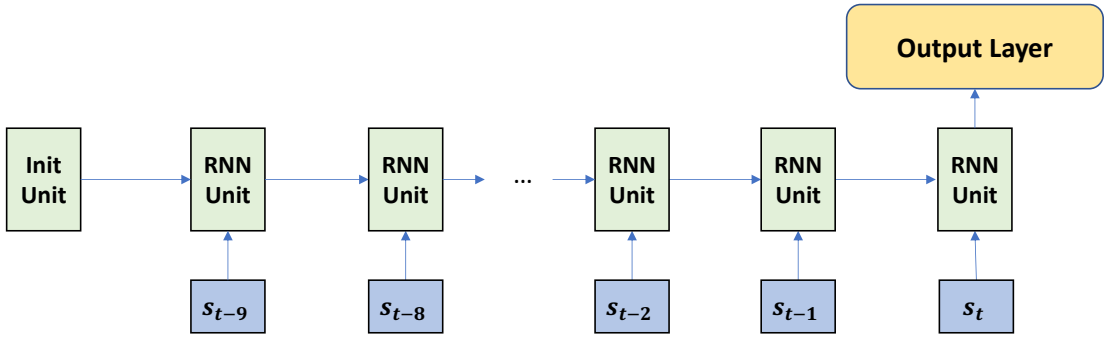


Figure 3.2: Recurrent Neural Network (RNN) Architecture

3.6 Numerical Experiment: Results

This section presents the results of the numerical experiments and discusses the interpretation of these results.

3.6.1 Best Performances

TD learning is applied to both the SL and RL method, with various update step m (see Section 3.3.4). These algorithms, SL-TD(m-step) and RL-TD(m-step), are trained using the training data, tuned with the validation data, and performances are reported using the testing data. Neural network architecture, learning rate, update step m, and other hyper-parameters are tuned to maximize the performance. The best performances using SL and RL are reported in Table 3.4. These figures are price gains per episode averaged over all 50 stocks. The price gain is reported in percentage of half-spread. The detailed performance for each stock can be found in Appendix (see Table A.3).

Given sufficient data and time, the RL method outperforms the SL method. This is true under both the stock-specific regime and the universal regime. The models trained under the universal regime generally outperforms the models trained under the stock-specific regime as well.

Price Gain (% Half-Spread)	**SL** (s.e.)	**RL** (s.e.)
Stock-Specific	21.40 (0.15)	24.82 (0.16)
Universal	22.34 (0.15)	25.47 (0.16)

Table 3.4: The universal model outperform the stock-specific models with both SL and RL by 4.4% and 2.6%, respectively. RL outperforms SL under the stock-specific and universal regime, by 16% and 14%, respectively. The figures reported are in units of percentage of half-spread (% half-spread).

3.6.2 Comparative Results

Both SL and RL method are specified by TD learning with various update step m (see Section 3.3.4). These TD specifications extend SL and RL method to two families of algorithms, SL-TD(m-step) and RL-TD(m-step). The update step m controls the target values of the neuron network

during training. Specifically, among T neurons in the output layer, m of them are matched to the empirical observations and $T - m$ are matched to the current model estimates. Different values of m and the difference between SL and RL presents various tradeoff in algorithm performance, which we will discuss shortly.

We will evaluate these algorithms using a few metrics, including their rate of convergence with respect to gradient steps, running time, their data efficiencies, and bias-variance tradeoff.

Rate of Convergence (Gradient Steps):

Figure 3.3 plots the price gain regression with respect to the number of gradient steps taken. We can see from the below figures, after controlling for the learning rate, batch size, neural network architecture, and other contributing factors, the RL method requires more gradient steps in SGD to converge compared to the SL method. It's also apparent that the convergence is slow when the update step m is small.

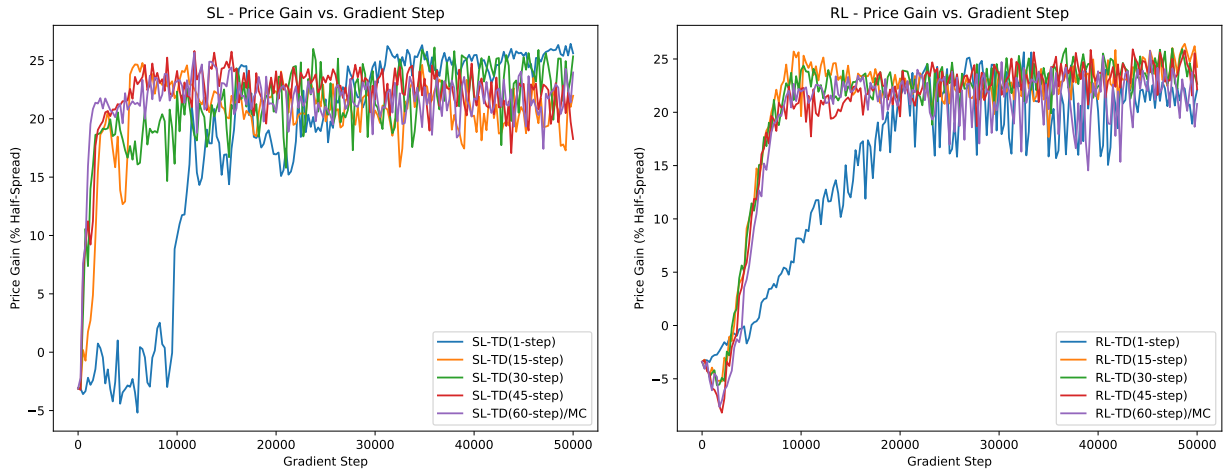


Figure 3.3: Price Gain vs. Gradient Step

Running Time:

Training neural networks can be time-consuming. Perhaps the most time-consuming part is iteratively taking gradient steps as part of the SGD procedure. Because a neural network typically takes thousands of steps to train, the time it takes to perform a single gradient step is an important

measurement to evaluate the running time of an algorithm. We will refer to this time as the gradient step time.

We have measured the average gradient step time over 50,000 gradient steps in SL-TD(m-step) and RL-TD(m-step). The result is plotted in Figure 3.4.

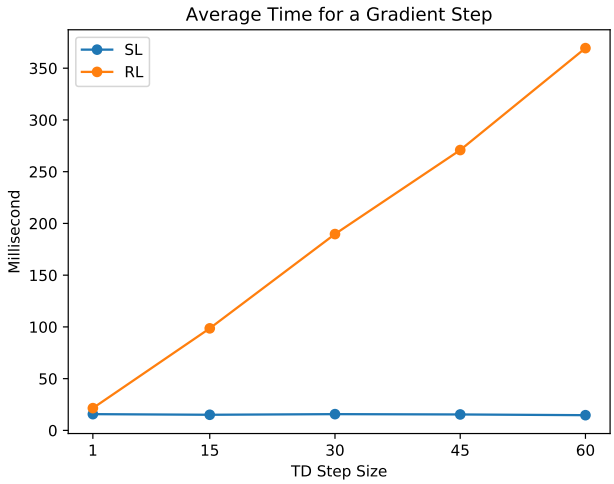


Figure 3.4: Average gradient step time (in milliseconds) averaged over 50k gradients steps. The standard error is negligible.

There are many factors that contributes to the gradient step time, such as the power of the CPU, the implementation choices and others. We have controlled all these factors so that the differences displayed in Figure 3.4 is solely due to the differences in TD step size m. The actual value of the gradient step time is rather irrelevant, however, it's clear that the gradient step time increases as the step size m increases in RL method, but stays flat in SL method.

This difference between SL and RL method comes down to the difference in the loss functions. In the SL method, in order to compute the the loss function for a specific data observation, it requires two neural network evaluations, namely $u^{\phi}(x_0)$ and $u^{\phi'}(x_m)$. This is true for all SL-TD(m-step), except for SL-TD(60-step). In SL-TD(60-step), (3.8) only evaluates the train-net once. This explains why gradient step time is relatively constant for different values of m in SL-TD(m-step).

On the other hand, in RL-TD(m-step), computing the loss function requires m neural network evaluations, which scales linearly with m. This can be seen in (3.33). This explains why gradient

step time roughly scales proportionally with the m in RL-TD(m-step).

Figure 3.5 plots the price gains progression with respect to elapsed running time. Among RL-TD(m-step), RL-TD(1-step) converges slowest with respect to gradient steps (see right figure in Figure 3.3). However, because each gradient step takes much less time, RL-TD(1-step) actually converges fastest in term of running time among all RL methods. In other words, given a fixed limited amount of time, RL-TD(1-step) achieves the best performance within all RL methods.

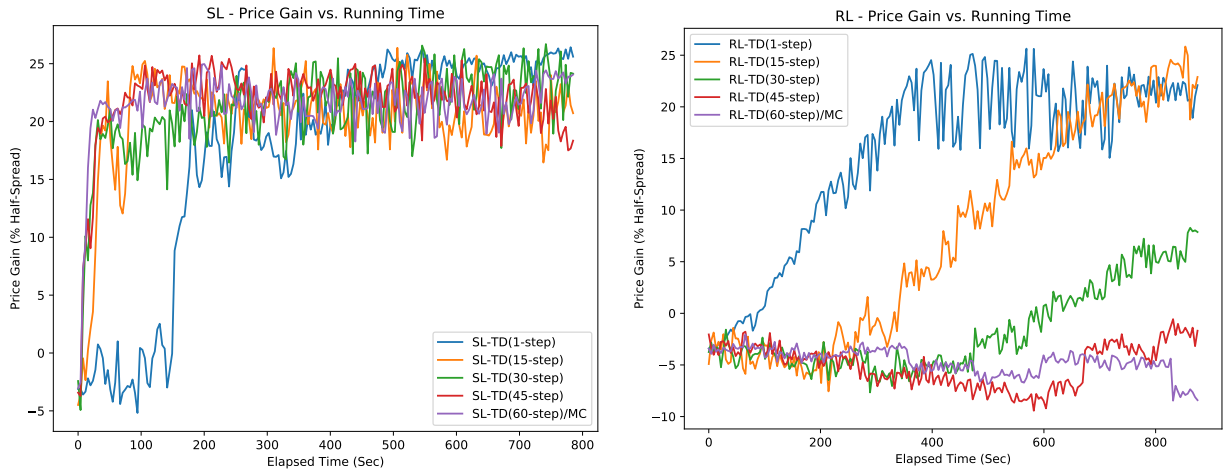


Figure 3.5: Price Gain vs. Running Time

Data Efficiency:

The central idea of TD learning is to use current model estimates instead of actual data observations to train models. Naturally, with different step sizes, TD method uses data differently. For SL method, the data required for a gradient step is (3.16) and for RL method, the data required is (3.34). For SL-TD(m-step) and RL-TD(m-step), it takes m time instances of data observations to perform a gradient step. At the extreme, TD(60-step) takes 60 times as much data to take a single gradient step than that of TD(1-step), in either the SL method or the RL method.

One way to evaluate the data efficiency of an algorithm is to evaluate its performance based on how much data it has accessed, measured in time instances. Figure 3.6 plots the price grain progression with respect to data accessed. It shows that TD(1-step) is the most data-efficient, in either SL or RL method. In other words, with the limited amount of data, TD(1-step) achieves the

best performance.

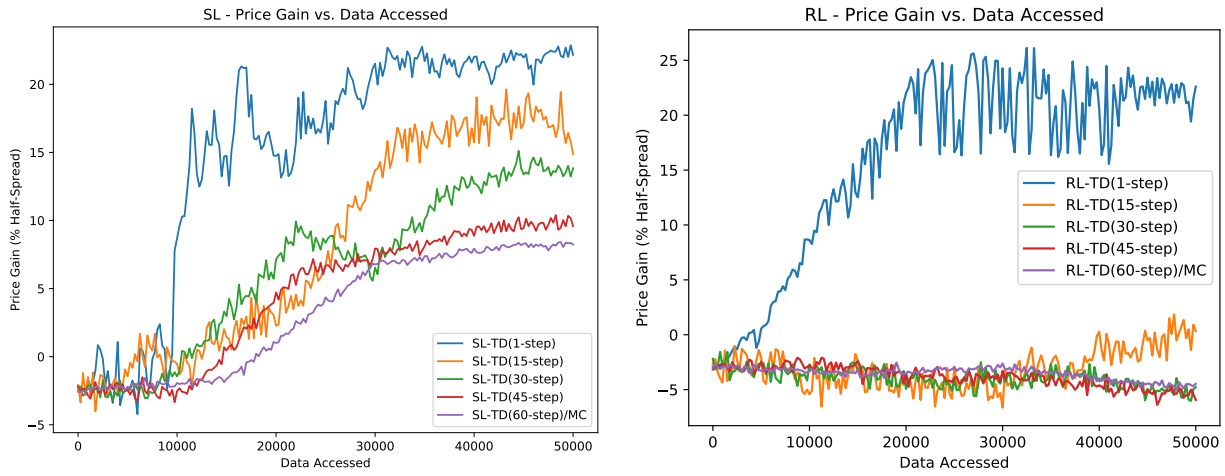


Figure 3.6: Price Gain vs. Data Accessed

Bias–Variance Tradeoff:

The bias–variance tradeoff has been a recurring theme in machine learning, and it's especially relevant in a discussion of TD learning. Previous studies have reported that TD update generally leads to higher bias and lower variance compared to Monte Carlo update when applied to the same prediction model (see Kearns and Singh 2000 and Francois-Lavet et al. 2019). We observe a similar pattern in our experiment.

As part of the SL method, the neural network is used to predict price change trajectories given an observable state variable. Consider a particular state x_0, and let $f_i(x_0)$ be the true price change at the ith time instances ahead. Then the price change trajectory can be represented as a vector of price changes $f(x_0) = [f_1(x_0), f_2(x_0), ..., f_{60}(x_0)]$. Let y_i be the observable price change at the ith time instance. Then $y_i = f_i(x_0) + \epsilon_i$, and the observable price change trajectory is $y = [y_1, y_2, ..., y_{60}]$.

Consider a set of training datasets, $\mathcal{D} = \{D_1, D_2, ..., D_n\}$. A neural network can be trained on each training dataset and produce a predicted trajectory in the SL method, denoted by $\hat{f}(x_0; D) = [\hat{u}_1(x_0; D), \hat{u}_2(x_0; D), ..., \hat{u}_{60}(x_0; D)]$. Averaging all these predictions from each dataset give the average ith price change prediction $\bar{u}_i(x_0) = \frac{1}{n}\sum_{i=1}^{n} \hat{u}_i(x_0; D_i)$ and the average price change trajectory prediction $\bar{f}(x_0) = \frac{1}{n}\sum_{i=1}^{n} \hat{f}(x_0; D_i)$. We now arrive at the following bias variance decomposition

for the prediction of the ith interval:

$$\text{MSE}_i(x_0) = \mathbb{E}_{D\in\mathcal{D}}\left[(y_i - \hat{u}_i(x_0; D))^2\right] \tag{3.37}$$

$$= \epsilon_i^2 + [f_i(x_0) - \bar{u}_i(x_0)]^2 + \mathbb{E}_{D\in\mathcal{D}}\left[(\bar{u}_i(x_0) - \hat{u}_i(x_0; D))^2\right] \tag{3.38}$$

$$= [y_i - \bar{u}_i(x_0)]^2 + \mathbb{E}_{D\in\mathcal{D}}\left[(\bar{u}_i(x_0) - \hat{u}_i(x_0; D))^2\right]. \tag{3.39}$$

Equation (3.38) is the common bias–variance decomposition, where ϵ_i^2 is the irreducible noise variance, $[f(x_0) - \bar{u}(x_0)]^2$ is the squared bias term, and $\mathbb{E}_{D\in\mathcal{D}}\left[(\bar{u}(x_0) - \hat{u}(x_0; D))^2\right]$ is the prediction variance. This decomposition can be reformulated as (3.39). Each term in (3.39) is observable and thus can be measured empirically. We will refer to $[y_i - \bar{u}_i(x_0)]^2$ as noise variance squared bias (NVSB).

A set of 100 training datasets are used, each producing a unique neural network. Testing these neural networks on the same testing dataset produces MSE, prediction variances, and NVSB for each time instances. A square root is taken on these values to obtain RMSE, the prediction standard deviations, and the noise standard deviation bias (NSDB). These values are averaged across all time instances and plotted in Figure 3.7.

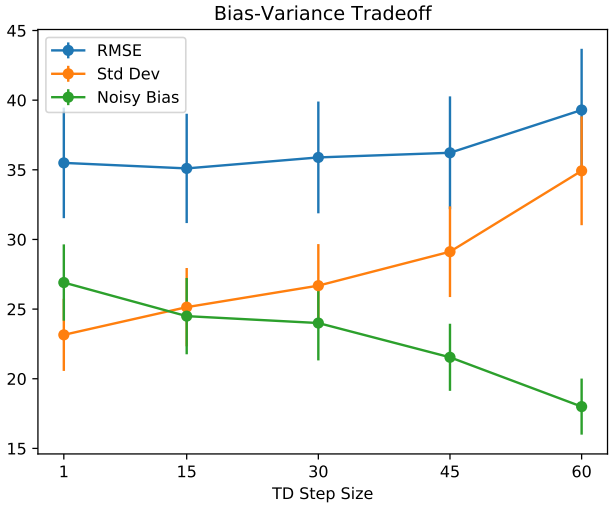


Figure 3.7: Bias–Variance Tradeoff

It's clear that there is a bias-variance tradeoff — TD with a smaller step size reduces variance and increases bias, and TD with a larger step size increases variance and reduces bias. A large prediction variance typically leads to overfitting. Indeed, this can also be observed empirically. When training the SL method using a small training dataset, the in-sample RMSE of TD(60-step) decreases quickly while its out-of-sample RMSE increases (see Figure 3.8). This is because TD(60-step) fits to the noisy patterns in the training data that don't generalize out of sample. Using the same training and testing data, TD(1-step) and TD(30-step) don't overfit nearly as much as TD(60-step).

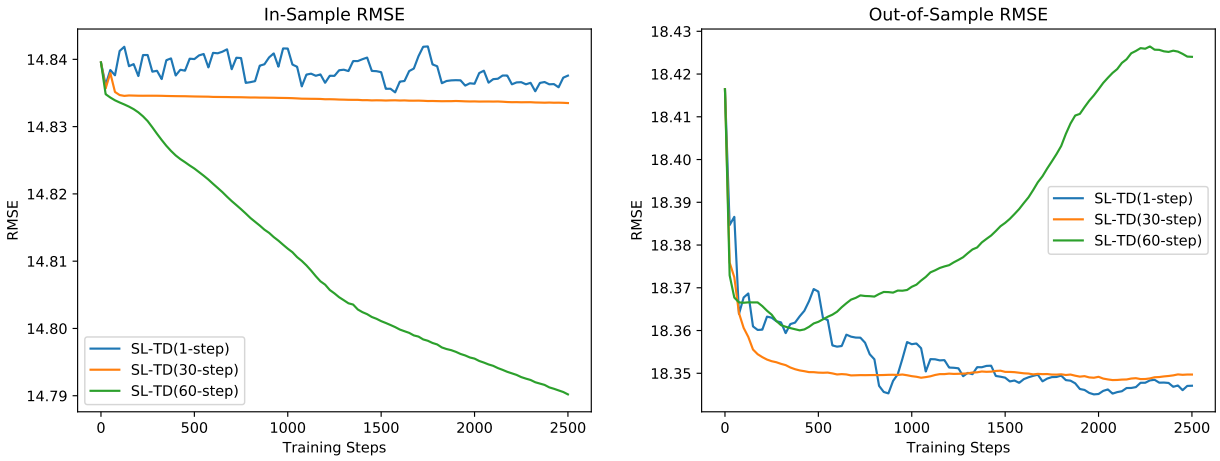


Figure 3.8: Left: In-Sample RMSE; Right: Out-of-Sample RMSE.

3.6.3 Universality

In Section 3.6.1, the universal models outperforms the stock-specific models. This reveals certain universality across stocks, that is, the experience learned from one stock can be generalized to a different stock. To further reinforce the evidence of this universality, we conduct another experiment under the "take-one-out" regime. Under the "take-one-out" regime, a model is trained on 49 stocks and tested on the stock that has been left out of the training. This way, the reported testing performance is out of sample in the conventional machine-learning sense and also on a stock that isn't part of the training data.

Table 3.5 displays the average performance of models trained under all three regimes. The

detailed performance for each model can be found in Appendix (see Table A.3). The take-one-out models performance comparable to the stock-specific models, indicating evidence of universality across stocks. However, the best performing model is still the universal model. This implies that there are still values in specific stocks.

Price Gain (% Half-Spread)	**SL** (s.e.)	**RL** (s.e.)
Stock-Specific	21.40 (0.15)	24.82 (0.16)
Take-one-out	21.55 (0.15)	24.85 (0.16)
Universal	22.34 (0.15)	25.47 (0.16)

Table 3.5: Performance comparison among models trained under all three regimes.

3.6.4 Result Summary

There isn't a single algorithm that is the most superior in all aspects. Rather, different algorithms might be preferable under different situations. The following lists some of these insights determined through the numerical results:

- Max Performance:
  - The RL method outperforms the SL method.
  - Universal model outperforms stock-specific model.

  If data and time aren't binding constraints and the goal is to maximize the performance, the universal RL model performs the best and is recommended for this situation.

- Time Limitation:
  - SL Method: Monte Carlo update method is fastest in convergence.
  - RL Method: TD(1-step) update method is fastest in convergence.

  If time is the binding constraint, then a fast algorithm is preferable. For the SL method, Monte Carlo update method (SL-TD(T-step)) is fastest with respect to running time. For the RL method, TD(1-step) provides the fastest convergence with respect to running time.

- Data Limitation:
  - SL Method: TD(1-step) update method is most data-efficient.
  - RL Method: TD(1-step) update method is most data-efficient.

  If the amount of data is the binding constraint, then a data-efficient algorithm is preferable. TD(1-step) provides the most data-efficient algorithms, for both SL method and the RL method.

- Prevent Overfitting:

  Monte Carlo update method leads to a high-variance and low-bias prediction model, which is prone to overfitting. TD learning leads to a low-variance and high-bias prediction, which provides the benefit of preventing overfitting.

Chapter 4: Variational Autoencoder for Risk Estimation

4.1 Introduction

Linear factor models (LFMs) are latent variable models that use unobservable variables to explain the variations in high-dimensional observable variables. In each a model, each observable variable is a linear combination of unobservable variables plus idiosyncratic noise. The unobservable variables are typically referred to as "factors," and are often of lower dimensionality than the observable variables.

Fitting a linear factor model to data is typically done through maximum likelihood estimation (MLE). If the number of factors is known and the idiosyncratic noises have uniform variance and are uncorrelated, then principal component analysis (PCA) provides an optimal parameter estimation for linear factor models. In such models, factors and observable variables are related through linear functions. This leads to certain restrictions on the distribution of the observable variables. Specially, the distribution of the observable variables follow a multivariate Gaussian distribution with a pre-specificed structure for the covariance matrix.

Variational autoencoders (VAEs) relax the linearity assumption in linear factor models, consequently, relax the Gaussian restrictions. VAEs utilize neural networks to model the relationship between factors and observable variables. This allows more general relationships between factors and observable variables, but inevitably leads to complications in model estimation. Because the likelihood can no longer be computed directly, the MLE method can no longer be used. Instead, to estimate parameters from data, VAEs use stochastic gradient descent (SGD) to maximize evidence lower bound (ELBO).

In this chapter, we make the connection between linear factor models and VAEs and argue that VAEs can be viewed as nonlinear factor models. Firstly, we show that linear factor models can be

formulated as linear VAEs — VAEs with linear functions instead of neural networks. The MLE provided by principal componential analysis can be shown to be optimal for a class of more general linear VAEs as well.

One of the applications of these models is modeling the covariance matrix of asset returns. The covariance matrix of asset returns is particularly difficult to estimate from historical data. This is mainly due to two reasons. One is that covariance matrix contains a lot of parameters, especially when the number of assets is large. When the number of assets is n, the number of parameters in the covariance matrix is of the order of n^2. The other reason is due to the fact that asset returns are time-varying. Historical returns from from the distant past is not an accurate reflection of the future return. Therefore, when predicting covariance matrix for the future asset return, the model should only incorporate historical data in the recent past. This limits the amount of the data that can be used. One way to address these difficulties is to use models such as linear factor models and VAEs. These models impose pre-specified structure on the covariance matrix, making the estimation more data efficient.

In finance, an important application of the covariance matrix is to help construct global minimum variance portfolio. To test the accuracy of the covariance matrix estimates, we construct minimum variance portfolios out-of-sample and compute the realized volatilities. We can show a moderate improvement from the covariance matrix estimate produced by VAEs over that of linear factor models. Another benefit of VAEs is their flexible structures. This allows us to incorporate side information into the covariance matrix estimate. Specially, we incorporate earnings data to dynamically adjust the covariance estimate, and this is proven to improve the minimum variance portfolios even further.

The rest of the chapter is organized as follows. Section 4.2 introduces minimum variance portfolios and their connections to asset return covariance. Section 4.3 introduces linear factor models and its maximum likelihood estimation via PCA. Section 4.4 introduces VAEs, and explains its relationships with LFMs. Section 4.5 explains the setup of numerical experiment and presents the results. Section 4.6 recap and conclude the chapter.

4.2 Application: Minimum Variance Portfolio

In finance, an important application of asset return covariance matrix is to evaluate the volatility of the return a portfolio of assets. Let $x \in \mathbb{R}^n$ be a random vector that represents the returns of n assets. The ith entry of this vector x_i represents the return of the ith asset. A portfolio of these n assets can be mathematically represented by $w \in \mathbb{R}^n$, a vector of portfolio weights — w_i represents the percentage of the capital allocated to the ith asset. The sum of the capital allocated to all assets is simply the total capital, in other words, $w^\top 1 = 1$. This is sometimes referred as "the budge constraint."

Given the asset return vector x and the portfolio weights w, the return of the portfolio is simply a linear combination of the return of each assets with the portfolio weights as coefficients, or mathematically,

$$x_w \triangleq \sum_{i=1}^{n} w_i x_i = w^\top x. \tag{4.1}$$

Let $\Sigma \in \mathbb{R}^{n \times n}$ be the covariance matrix of asset returns, in other words, for any $1 \leq i, j \leq n$, $\Sigma_{ij} = \mathrm{Cov}(x_i, x_j)$. Now we can express the variance of the portfolio return as follow

$$\mathrm{Var}(x_w) = w^\top \mathrm{Cov}(x) w = w^\top \Sigma w. \tag{4.2}$$

The volatility of the portfolio return is simply the square root fo the variance, given by $\sigma(x_w) = \sqrt{w^\top \Sigma w}$. Different portfolio weights on the same group of assets could lead to very different portfolio volatility. To minimize the portfolio volatility, one would typically hold assets that have negatively correlated returns to offset the gains and losses, effectively achieving a hedge. This can be done for all the assets by systematically determine the portfolio weights to minimize the overall portfolio variance. This procedure can be formulated as the following optimization problem.

$$\min_{w} \quad w^\top \Sigma w \tag{4.3}$$

$$\text{s.t.} \quad w^\top 1 = 1 \tag{4.4}$$

Using Lagrangian, the optimal solution can be solved analytically, given by

$$w^* = \frac{\Sigma^{-1}1}{1^\top \Sigma^{-1}1}. \tag{4.5}$$

The resulting portfolio w^* is typically referred as the "minimum variance portfolio," as it seeks to achieve the lowest portfolio variance. The portfolio variance of the minimum variance portfolio is given by $w^{*\top}\Sigma w^*$, which is the lowest achievable portfolio variance while obeying the budget constraint.

These discussions above is based on a given asset covariance matrix Σ, however, in practice, Σ is typically unknown and needs to be estimated. The rest of this paper will discuss a few methods to estimate Σ from historical return data.

4.3 Linear Factor Model

This section introduces the general problem of estimating covariance matrices through maximum likelihood estimation. Linear factor models and their estimation procedures via principal component analysis are introduced.

4.3.1 Maximum Likelihood Estimation

We consider the problem of estimating covariance matrix in a Gaussian setting. Specially, assume that there is an underlying data generating distribution $x \sim N(0, \Sigma_*)$ with an unobservable covariance matrix $\Sigma_* \in \mathbb{R}^{n\times n}$.

We seek to choose an estimate $\hat{\Sigma}$ to approximate Σ_*. One way is to choose $\hat{\Sigma}$ to maximize the log-likelihood. Given a single data point $x^{(i)}$, the likelihood of observing this data point is

$$L\left(\hat{\Sigma}, x^{(i)}\right) \triangleq \log p\left(x^{(i)}|\hat{\Sigma}\right) = -\frac{1}{2}\left(n\log(2\pi) + \log\det(\hat{\Sigma}) + x^{(i)\top}\hat{\Sigma}^{-1}x^{(i)}\right), \tag{4.6}$$

where $p\left(x^{(i)}|\hat{\Sigma}\right)$ is the probability density function of $N(0, \hat{\Sigma})$ evaluated at $x^{(i)}$. When given a data set $X = \{x^{(1)}, x^{(2)}, ..., x^{(N)}\}$, one can simply choose an estimate $\hat{\Sigma}$ to maximize the average

log-likelihood

$$\log p(X|\hat{\Sigma}) \triangleq \frac{1}{N}\sum_{i=1}^{N} \log p\left(x^{(i)}|\hat{\Sigma}\right) = -\frac{1}{2}\left(n\log(2\pi) + \log\det(\hat{\Sigma}) + \mathrm{tr}(\hat{\Sigma}^{-1}\Sigma_{\mathrm{SAM}})\right), \tag{4.7}$$

where $\Sigma_{\mathrm{SAM}} = \dfrac{\sum_{i=1}^{N} x^{(i)}x^{(i)\top}}{N}$ is the sample covariance matrix. Observe that maximizing (4.7) is equivalent to minimizing the Kullback-Leibler divergence between $N(0,\hat{\Sigma})$ and $N(0,\Sigma_{\mathrm{SAM}})$. It can be shown that $\hat{\Sigma}_{\mathrm{MLE}} = \Sigma_{\mathrm{SAM}}$, in other words, the MLE estimator is the sample covariance matrix. However, sample covariance matrix performs poorly out-of-sample, especially in the case when the sample size N isn't much larger than the dimension n. One of the ways to address this issue is to impose factor structure, which we will discuss for the remainder of this section.

4.3.2 Linear Factor Model

Linear factor models relate a high dimensional vector x to a low dimensional vector z through a linear transformation.

$$x = Lz + \epsilon, \tag{4.8}$$

where $x \in \mathbb{R}^n$ is the observable variables, $z \in \mathbb{R}^k$ is the latent variables and is also commonly referred as "factors", and $L \in \mathbb{R}^{n\times k}$ is the factor loading matrix. The dimension of the factors k is treated as a hyper-parameter of the model and is typically much smaller than the dimension of the observable variables ($k << n$).

Linear factor models are typically assumed to have Gaussian priors and Gaussian noise. Specially, the distribution of z is typically set to be standard normal $z \sim N(0, I_k)$, where I_k is the identify matrix of size $k \times k$. This is also commonly referred to as "the prior of z." The idiosyncratic noise is assumed to be Gaussian noise, i.e. $\epsilon \sim N(0, \sigma^2 I_n)$, where I_n is the identify matrix of size $n\times n$. In order words, the idiosyncratic noise is assumed to be uncorrelated with each other and have the same variance σ^2. This is the simplest covariance structure of the idiosyncratic noises, and this is commonly referred as the "isotropic" case.

These distributional assumptions lead to the following conditional and marginal distribution of x,

$$x|z \sim N(Lz, \sigma^2 I_n), \tag{4.9}$$

$$x \sim N(0, LL^\top + \sigma^2 I_n). \tag{4.10}$$

Effectively, linear factor models imply a specific distribution on the data generating distribution, namely (4.10). Instead of estimating the covariance matrix directly, due to the structure imposed, we now only need to estimate the parameters L and σ. In other words, linear factor models is a way of achieving low-rank approximation.

4.3.3 Principal Component Analysis

Principal component analysis provides the maximum likelihood estimation for linear factor models. Specifically, for samples $X = \{x^{(1)}, x^{(2)}, ..., x^{(N)}\}$, we seek to solve the following optimization problem.

$$\max_{L \in \mathbb{R}^{n \times k}, \sigma^2 \in \mathbb{R}_+} \quad \log p(X|\hat{\Sigma}) \tag{4.11}$$

$$\text{s.t.} \quad \hat{\Sigma} = LL^\top + \sigma^2 I_n \tag{4.12}$$

According to Tipping and Bishop 1999, the optimal solution is given by the principal component analysis. This involves the following procedure.

1. Compute the sample covariance matrix $\Sigma_{\text{SAM}} = \dfrac{\sum_{i=1}^N x^{(i)} x^{(i)\top}}{N}$;
2. Use eigenvalue decomposition to decompose sample covariance matrix $\Sigma_{\text{SAM}} = U \Lambda U^{-1}$, where $U = [u_1 ... u_n]$ and $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$ with $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$;
3. Compute $\hat{\sigma}^2 = \dfrac{1}{n-k} \sum_{i=k+1}^n \lambda_i$ and $\hat{L} = U_k (\Lambda_k - \hat{\sigma}^2 I_k)^{1/2} R$, where $U_k = [u_1 ... u_k]$, $\Lambda_k = \text{diag}(\lambda_1, ..., \lambda_k)$, and R is an arbitrary orthogonal rotation matrix.

In other words, the estimate for the residual variance $\hat{\sigma}^2$ equals the average of the smallest $n - k$

eigenvalues of Σ_{SAM}, which has a clear interpretation as the average variances unexplained by the dimensions spanned by the top k eigenvectors. The estimate for the factor loading matrix is a linear combinations of the top k eigenvectors associated with the largest k eigenvalues. Note that $\hat{L}$ can't be identified uniquely as R can be any orthogonal rotational matrix. However, conventionally, R is simply ignored (or $R = I_k$) for simplicity. These PCA estimates lead to an estimate for the covariance matrix, which we denote as $\hat{\Sigma}_{\text{PCA}} = \hat{L}\hat{L}^\top + \hat{\sigma}^2 I_n$.

Linear factor models assume a linear relationship between the factors z and the observable variables x. The linearity ensures that the observable variable x also follows a Gaussian distribution with a specific covariance matrix structure. One way to extend this model is to relax the linearity assumption and allow more complex relationships between z and x. This would lead to a more general distribution of x. In the next section, we will discuss how this can be achieved via variational autoencoders.

4.4 Variational Autoencoders

Similar to linear factor models, variational autoencoders (VAEs) also use latent variables (or factors) z to model the distribution of observable variables x. The main difference, however, is that VAEs use neural networks to model the relationship between z and x instead of a linear function. This allows VAEs to model much more general distributions than linear factor models.

This section introduces the general framework of latent variable models and details of VAEs including model primitives and estimation procedures. The relationship between linear factor models and VAEs is also discussed.

4.4.1 Latent Variable Models

Latent variable models are a class of models that uses latent variables to model the distribution of observable variables. The goal of these models is to estimate the data generating distribution $p(x)$ for observable variables x, defined over a potentially high dimensional space $\mathcal{X}$. Latent variables are used to impose structure on $P(x)$.

More formally, let z be a vector of latent variables in a high-dimensional space $\mathcal{Z}$ and follow a probability density function $P(z)$ defined over $\mathcal{Z}$. This distribution is often referred to as "the prior of z." Let's also denote the factor transformation function $f_\theta : \mathcal{Z} \to \mathcal{X}$ be a family of deterministic functions parameterized by $\theta \in \Theta$. We can then write the relationship between x and z as follows

$$x = f_\theta(z) + \epsilon. \tag{4.13}$$

The conditional distribution $P(x|z)$ is called the output probability distribution. This distribution depends on $f_\theta(z)$ and the distributional assumption on the noise term ϵ. Assuming isotropic Gaussian noise, i.e. $\epsilon \sim N(0, \sigma^2 I_n)$, we can specify the output probability distribution

$$P_{\theta,\sigma}(x|z) \sim N\left(f_\theta(z), \sigma^2 I_n\right). \tag{4.14}$$

This allows us to represent the data generating distribution as follows

$$P(x) = \int P_{\theta,\sigma}(x|z)P(z)dz. \tag{4.15}$$

Given an observable data set $X \in \mathcal{X}$, the goal is to find the optimal θ such that the resulting data generating process is most likely to have generated the data set X, or equivalently (4.15) achieves maximum.

Linear factor models are a class of latent variable models as well. The deterministic function $f_\theta(z)$ is replaced by a simple linear function $f_L(z) = Lz$ with L being the parameter, and the output probability distribution is $P_{L,\sigma}(x|z) = N(Lz, \sigma^2 I_n)$. The likelihood function $P(x)$ is $N(0, LL^\top + \sigma^2 I_n)$, which doesn't need to be computed using (4.15), but can be derived directly using Gaussian properties.

4.4.2 Variational Autoencoder

In VAEs, $f_\theta(z)$ is typically modeled using a neural network, with θ being the neural network parameters. The output probability distribution is often chosen to be Gaussian, i.e.

$$P_{\theta,\sigma}(x|z) \sim N(f_\theta(z), \sigma^2 I_n).$$

With $f_\theta(z)$ being a general neural network, (4.15) can not be solved analytically, and therefore $P(x)$ becomes intractable. This prevents us to maximize the likelihood of observing a data set directly. The posterior $P_{\theta,\sigma}(z|x) = \dfrac{P_{\theta,\sigma}(x|z) \cdot P(z)}{P(x)}$ is also intractable.

4.4.3 Estimation via Evidence Lower Bound

A important breakthrough in VAEs is the introduction of another neural network $g_\phi(x)$, where ϕ is the neural network parameters, to specify an approximate posterior distribution $Q(z|x)$. For computational simplicity, this approximate posterior is typically specified as a Gaussian with independent noise terms

$$Q_{\phi,\eta}(z|x) \sim N\left(g_\phi(x), \text{diag}(\eta)\right), \tag{4.16}$$

where $g_\phi(x)$ is the deterministic function modeled by the neural network, and $\text{diag}(\eta)$ is a diagonal matrix with a non-negative vector η as the diagonal.

Because the log-likelihood can't be computed and maximized directly, VAEs use a different procedure to estimate parameters. This procedure hinges on the following identity. Given a data point $x^{(i)}$, the following holds.

$$\log P(x^{(i)}) - \mathcal{D}_{KL}\left[Q(z|x^{(i)})||P(z|x^{(i)})\right] = \mathbb{E}_{z\sim Q}\left[\log P(x^{(i)}|z)\right] - \mathcal{D}_{KL}\left[Q(z|x^{(i)})||P(z)\right]. \tag{4.17}$$

The right side of (4.17) is also called evidence lower bound (ELBO) as it is a lower bound of

the log-likelihood. Specifically, because $\mathcal{D}_{KL}\left[Q(z|x^{(i)})||P(z|x^{(i)})\right] \geq 0$ for any data point $x^{(i)}$,

$$\mathcal{L}_{ELBO}(x^{(i)}) \triangleq \mathbb{E}_{z\sim Q}\left[\log P(x^{(i)}|z)\right] - \mathcal{D}_{KL}\left[Q(z|x^{(i)})||P(z)\right] \leq \log P(x^{(i)}) \tag{4.18}$$

Because every component in the ELBO is tractable, we can maximize the ELBO instead of the log-likelihood. This leads to an interesting interpretation. Due to the equality in (4.17), maximizing the ELBO is equivalent to maximizing $\log P(x^{(i)})$, the log-likelihood of observing the data point, while minimize $\mathcal{D}_{KL}\left[Q(z|x^{(i)})||P(z|x^{(i)})\right]$ simultaneously, the KL-divergence of the true posterior and the approximate posterior. The KL-divergence term can be interpreted as a regularization term that forces $Q(z|x^{(i)})$ to be close to $P(z|x^{(i)})$. If the true posterior $P(z|x^{(i)})$ can be expressed by the approximate posterior $Q(z|x^{(i)})$, then the ELBO is a binding lower bound.

To summarize, a VAE is specified by its output probability and approximate posterior, which typically take on the following form

$$\begin{cases} P_{\theta,\sigma}(x|z) \sim N\left(f_\theta(z), \sigma^2 I\right) \\ Q_{\phi,\eta}(z|x) \sim N\left(g_\phi(x), \text{diag}(\eta)\right). \end{cases} \tag{4.19}$$

The parameters that needs to be estimated from the data is $(\theta, \sigma, \phi, \eta)$. The estimation procedure involves running SGD to maximize ELBO. This can be written as the following optimization problem (X is the observed data set).

$$\begin{aligned} \max_{\theta,\sigma,\phi,\eta} \quad & \mathbb{E}_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}_{KL}\left[Q(z|X)||P(z)\right] \\ \text{s.t.} \quad & P_{\theta,\sigma}(x|z) \sim N\left(f_\theta(z), \sigma^2 I\right) \\ & Q_{\phi,\eta}(z|x) \sim N\left(g_\phi(x), \text{diag}(\eta)\right) \\ & \eta \geq 0 \end{aligned} \tag{4.20}$$

4.4.4 Covariance Estimation

In the settings of Section 4.2 where the goal is to estimate the covariance matrix, we can simulate from the trained VAE to estimate covariance. Using the law of total covariance, we can derive the covariance matrix from VAEs are as follow.

$$\Sigma_{\text{VAE}} \triangleq \text{Cov}(x) = \text{Cov}(f_\theta(z)) + \sigma^2 I_n \tag{4.21}$$

The simulation procedure to obtain an estimate of Σ_{VAE} is as follows.

1. Simulate $\{z^1, ..., z^N\}$ from $P(z)$.
2. Compute $f^{(i)} = f_\theta\left(z^{(i)}\right)$ for each $z^{(i)}$ and obtain $\{f^{(1)}, ..., f^{(N)}\}$.
3. Compute the sample covariance matrix for $\{f^{(1)}, ..., f^{(N)}\}$. This is $\hat{\text{Cov}}(f_\theta(z)) = \dfrac{\sum_{i=1}^N f^{(i)} f^{(i)\top}}{N}$.
4. Compute $\hat{\Sigma}_{\text{VAE}} = \hat{\text{Cov}}(f_\theta(z)) + \sigma^2 I_n$.

4.4.5 Linear Factor Models as Variational Autoencoders

Linear factor models can be reformulated as a linear VAE. The distributional assumptions of linear factor models lead to the output probability and likelihood function directly as follows.

$$\begin{cases} P_{L,\sigma}(x|z) \sim N(Lz, \sigma^2 I) \\ P(x) \sim N(0, LL^\top + \sigma^2 I_n) \end{cases} \tag{4.22}$$

Due to the linearity of the model, the posterior of z can be computed analytically using Bayes' theorem. Lemma 4.4.1 states this more formally.

Lemma 4.4.1. *Given a linear factor model*

$$x = Lz + \epsilon; \quad \epsilon \sim N(0, \sigma^2 I_n), \tag{4.23}$$

with the prior $z \sim N(0, I_k)$, the posterior of z given x is given by

$$P(z|x) \sim N\left(\sigma^{-2} S L^\top x, S\right), \tag{4.24}$$

where $S = (I_k + \sigma^{-2} L^\top L)^{-1}$

The proof of this lemma is presented in Appendix (see Proof B.1). Lemma 4.4.1 allows us to formulate VAEs that are equivalent to linear factor models.

Equivalent VAE:

Consider the following VAE with linear functions instead of neural networks. The set of parameters in this VAE that needs to be estimated is (L, σ).

$$\begin{cases} P_{L,\sigma}(x|z) \sim N(Lz, \sigma^2 I) \\ Q_{L,\sigma}(z|x) \sim N(\sigma^{-2} S L^\top x, S), \end{cases} \tag{4.25}$$

Consider the ELBO of this linear VAE, because $Q_{L,\sigma}(z|x)$ is set to be the same as $P(z|x)$, $\mathcal{D}_{KL}\left[Q(z|x)||P(z|x)\right] = 0$. This implies that the ELBO coincides with the log-likelihood. In order words, for any data set X, the following is true for the VAE in (4.25).

$$\mathcal{L}_{\text{ELBO}}(X) = \mathbb{E}_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}_{KL}\left[Q(z|X)||P(z)\right] = \log P(X) \tag{4.26}$$

As a result, maximizing ELBO is equivalent to maximizing the log-likelihood directly. Therefore, the PCA estimates of (L, σ) is an optimal solution to the estimation problem in (4.20) for the specific VAE in (4.25). In other words, this linear VAE is equivalent to a linear factor model. The following theorem is immediate.

Theorem 4.4.2. *The variational autoencoder represented in* (4.25) *is equivalent to a linear factor model with parameter* (L, σ)*. The equivalence is in the sense that the parameters, objective functions and optimal solutions are the same between the two models.*

General Linear VAEs:

The linear VAE with (4.25) is essentially a reformulation of a linear factor model. This shows that linear factor models conform to the VAE frameworks and can be viewed as a special case of a linear VAE. However, this linear VAE is somewhat degenerate. In (4.25), we purposefully set $Q_{L,\sigma}(z|x)$ to be the same as the true posterior $P(z|x)$. This makes the $Q_{L,\sigma}(z|x)$ redundant — if we can compute the true posterior, then we don't need to construct $Q_{L,\sigma}(z|x)$ to approximate it. Also, $Q_{L,\sigma}(z|x)$ doesn't have any free parameters — its value is entirely dictated by L and σ, the same parameters in $P_{L,\sigma}(x|z)$.

Consider a more general VAE with linear decoding process but general encoding process given by

$$\begin{cases} P_{L,\sigma}(x|z) \sim N(Lz, \sigma^2 I) \\ Q_{\phi,\Sigma}(z|x) \sim N(g_\phi(x), \Sigma). \end{cases} \tag{4.27}$$

The encoding function $g_\phi(x)$ can be any mapping from $\mathcal{X}$ to $\mathcal{Z}$ and Σ can be any positive semi-definite matrix. In this VAE, the encoding and decoding processes are also decoupled, with its own parameters. The parameters in this model are $(L, \sigma, \phi, \Sigma)$.

In this VAE, we can show that the PCA solution and its associated posterior is still an optimal solution to the VAE. Theorem 4.4.3 states this more formally. The proof is deferred until Appendix B.2.

Theorem 4.4.3. *Given a data set X, let the PCA estimate of a linear factor model be* $\hat{L}$ *and* $\hat{\sigma}$. *Define* $(L^*, \sigma^*) \triangleq (\hat{L}, \hat{\sigma})$, $g^*_\phi(x) \triangleq \hat{\sigma}^{-2} S \hat{L} x$, *and* $\Sigma^* = S$ *where* $S = (I_k + \hat{\sigma}^{-2} \hat{L}^\top \hat{L})^{-1}$. *This set of parameters* $(L^*, \sigma^*, g^*, \Sigma^*)$ *is a set of optimal parameter estimates for the VAE represented in* (4.27).

4.5 Numerical Experiments

In our numerical experiments, the linear factor models and variational autoencoders are used to estimate the covariance matrix of high-dimensional stock return data. Their estimation accuracy

is compared using various metrics. Sample covariance matrix can be computed and used as a benchmark. Minimum variance portfolios can be constructed based on each covariance estimate, and their realized volatility is compared as well. This section outlines the setup and the results of these numerical experiments.

4.5.1 Stock Return Data

The experiments are conducted using historical daily returns of S&P500 constituents as of January 1st 2012. The daily returns are collected for the subsequent 6 years, from January 1st 2012 to December 31st 2017. The stocks that were delisted or have degenerate returns are removed. This leaves us with 429 stocks and 1509 daily return observations. This produces a data set of raw returns $R = \{r^{(1)}, ..., r^{(1509)}\}$, where each daily return vector $r^{(t)} \in \mathbb{R}^{429}$ for all $1 \leq t \leq 1509$.

The raw return is then standardized by its trailing mean and standard deviation. In other words, let $r_i^{(t)}$ be the return of ith stock on day t, then let the standardized return $x_i^{(t)}$ as follow

$$x_i^{(t)} = \frac{r_i^{(t)} - \mathbb{E}[r_i^{(t)}]}{\sigma_i^{(t)}}. \tag{4.28}$$

The mean $\mathbb{E}[r_i^{(t)}]$ and standard deviation $\sigma_i^{(t)}$ are estimated using their empirical counterparts estimated from the prior 100 days' observations. This transforms the raw return R to the standardized return X.

In order to accurately report the estimation results out-of-sample, the standardized return data is divided into training, validation and testing data sets, which consists of 1000, 20 and 20 daily return observations respectively. Models are trained on the training data, tuned to optimize performance on the validation data, and the results are reported on the testing data set. In the case of linear factor models, only the dimension of factor k is tuned. In the case of VAEs, the dimension of factor k and the neural network architecture are turned. This procedure is then repeated with train, validate, and test data advancing 20 days each. This rolling window of training, validating and testing is continued until the end of the data period.

4.5.2 Estimation Accuracy Comparison

One way to evaluate the accuracy of the covariance matrix estimate is through their log-likelihood on the observed data. Three estimates are compared using this metric — sample covariance matrix, linear factor model (PCA estimate), and VAE estimate. Because the log-likelihood is intractable in the VAE, the value of the evidence lower bound (ELBO) is used instead. The value of the ELBO and the log-likelihood is not a direct comparison, however, because the ELBO is a lower bound of the log-likelihood, it provides some insights regarding the log-likelihood of the VAE.

The results are reported on the test data set and therefore the results are out-of-sample. PCA estimate and VAE estimate are tuned to maximize their results on the validation set. Figure 4.1 plots the log-likelihood of sample covariance matrix and the PCA estimates, and the value of ELBO of the VAEs on the testing data throughout the data period.

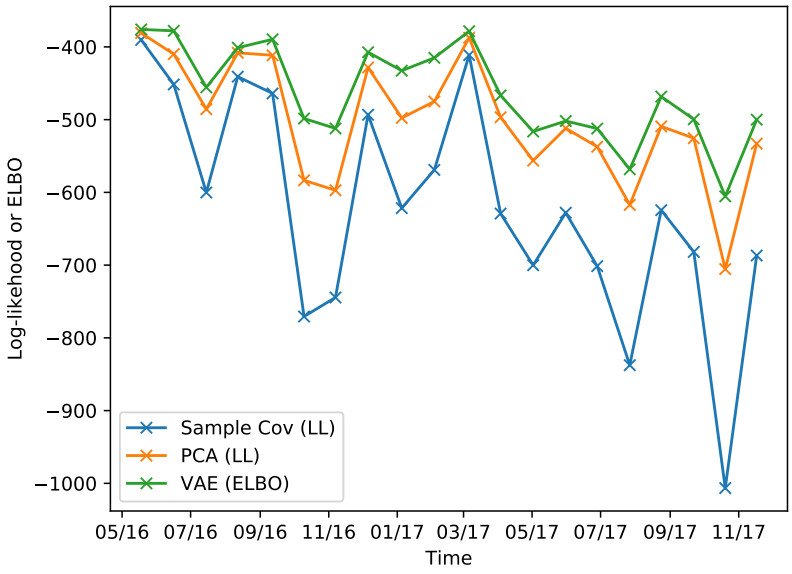


Figure 4.1: Covariance Matrix Estimate Comparison

Figure 4.1 shows that the sample covariance matrix has the lower log-likelihood value, indicating the worst out-of-sample performance among the three. The sample covariance matrix has no

structure imposed on the covariance matrix and therefore easily overfit to the training data. This leads to a poor out-of-sample performance.

The value of the ELBO of the VAEs are consistently higher than the log-likelihood of the PCA estimates. Because ELBO is a lower bound, the log-likelihood of the VAEs must be higher than the log-likelihood of the PCA estimates as well. This shows that VAEs produce a better fit to the data than that of the linear factor model.

Another notable difference between the PCA estimate and the VAE estimate is the optimal number of factors, or k — the dimension of the latent variable z. Figure 4.2 plots the average performance (the log-likelihood for PCA estimates and the ELBO for VAE estimates) across all testing data sets with different value of k. As can be seen from Figure 4.2, when k is very small, both model underfits the data and the out-of-sample performance is suboptimal. On the other hand, when k is too large, the models overfit the training data and the out-of-sample-performance deteriorates as well. The number of factors that yields the best out-of-sample performance is the optimal value for k.

The difference between the PCA estimates and the VAE estimates is their optimal value for k. The PCA estimates' performance peak at $k = 30$, whereas the VAE estimates' peak at $k = 10$. This aligns with our intuitions — comparing to the linear factor model, VAE is a more flexible model, therefore it takes less number of factors for it to overfit to the data.

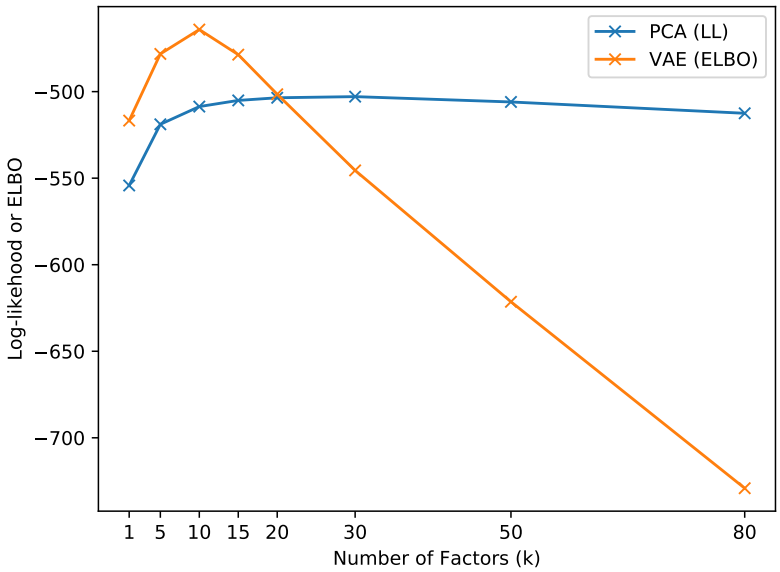


Figure 4.2: Number of Factors

4.5.3 Minimum Variance Portfolios

An important application of covariance matrix estimation is to construct minimum variance portfolios. In our experiments, this can be done on the standardized return X as well as the raw return R.

Minimum Correlation Portfolio:

Linear factor models and the VAEs are applied to estimate the covariance matrix of the standardized return, namely $\hat{\text{Cov}}(X)$. Because X is the standardized return, $\hat{\text{Cov}}(X)$ is essentially an estimate of the correlation matrix of the raw return R.

Given an estimate $\hat{\text{Cov}}(X)$, a minimum variance portfolio is given by

$$w^* = \frac{\hat{\text{Cov}}(X)^{-1}1}{1^\top \hat{\text{Cov}}(X)^{-1}1}.$$

The realized volatility of this portfolio throughout the testing period is used as a metric to

evaluate the accuracy of the covariance estimates. The volatility is measured using standardized returns of the portfolios. Table 4.1 displays the average realized volatility of a few portfolios throughout the entire testing period.

	Realized Volatility (Stand. Return)	reduct. %
Single Asset port. (Idealized)	1	-141%
Equal Weight port.	0.4145	0%
Min-Var port. (SAM)	0.4009	3%
Min-Var port. (PCA)	0.3309	20%
Min-Var port. (VAE)	0.296	29%

Table 4.1: Realized volatilities of various portfolios measured on standardized return data X. The reduction percentage (reduct. %) is relative to the equal weight portfolio.

Because standardized return X is constructed to have unit variance, holding any single asset would have average volatility very close to 1. The realized volatility of an idealized single asset portfolio would be 1, and this is meant to be a benchmark for other methods.

Equal weight portfolio allocates $\frac{1}{n}$ of the total capital to each of the n assets, namely $w_{\text{eq}} = \mathbf{1}/n$. This portfolio diversifies among all assets equally, without taking the correlation structure into the account.

Minimum variance portfolios can be constructed based on sample covariance (SAM), linear factor model (PCA), and the variational autoencoder estimate (VAE).

As we can see in Table 4.1, the effect of diversification is very significant — a simple equal weight portfolio reduces the realized volatility more than half compare to any single asset. The effect of minimizing portfolio variance is also significant — the minimum variance portfolio built on PCA estimate reduces 20% of the volatility compare to the equal weight portfolio, while the VAE estimate performs the best, yielding roughly another 10% reduction compare to the PCA estimate.

Minimum Variance Portfolio:

The experiments above are conducted on the standardized return X, effectively ignoring the time-varying volatility of each asset. To implement minimum variance portfolio that can be implemented in practice, we need to incorporate the stock volatilities as well.

Because standardized return X is constructed by normalizing the stock volatility, the covariance matrix of the raw return R can be computed based upon the estimates of the covariance of standardized return. More specifically, their relationship can be expressed mathematically as

$$\hat{\mathrm{Cov}}(r_t) = \mathrm{diag}(\hat{\sigma}_t)\hat{\mathrm{Cov}}(X)\,\mathrm{diag}(\hat{\sigma}_t), \tag{4.29}$$

where $\hat{\mathrm{Cov}}(X)$ is estimated using either linear factor models or the VAEs, and $\mathrm{diag}(\hat{\sigma}_t)$ is the diagonal matrix with return volatility σ_t on the diagonal, which is estimated using the standard deviations of trailing 100 days' return.

The minimum variance portfolio on the raw return during day t can be constructed as

$$w_t^* = \frac{\hat{\mathrm{Cov}}(r_t)^{-1}1}{1^\top \hat{\mathrm{Cov}}(r_t)^{-1}1}.$$

Because the return volatility estimate $\hat{\sigma}_t$ change everyday, the return covariance estimate $\hat{\mathrm{Cov}}(r_t)$ also change everyday, leading to a different daily portfolio w_t^*.

The realized volatility on this daily-updated minimum variance portfolio is used as a metric to evaluate various covariance estimate. Table 4.2 displays the results.

	Realized Volatility (Raw Return)	reduct. %
Equal Weight port.	10.38%	0
Min-Var port. (SAM)	6.58%	37%
Min-Var port. (PCA)	6.07%	42%
Min-Var port. (VAE)	5.91%	43%

Table 4.2: Realized portfolio volatilities on the raw return data R. The reduction percentage (reduct. %) is relative to the equal weight portfolio. Realized volatility is annualized.

4.6 Conclusion

Linear factor models and variational autoencoders are both latent variable models that models the distribution of high-dimensional random variable. In this chapter, we make the connection between these two classes of models.

Specifically, we show that a class of linear VAE is equivalent to linear factor models, and the PCA solution also provides the optimal parameter estimations to linear VAEs. From this, we can view nonlinear VAEs as an extension to linear factor models by relaxing the linearity assumption. This relaxation expands the class of distribution that the model can represent, and potentially enables the model to model data more accurately.

One of the applications of linear factor models and VAEs is to approximate asset return covariance. The asset return covariance plays an important role in portfolio construction, that is, the volatility of a portfolio depends on the covariance matrix of the individual asset returns. However, the covariance matrix is typically unknown and needs to be approximated from historical data. This is a generally a difficult task, mainly due to the time-varying nature of asset returns and the high-dimensionality of the covariance matrix. Linear factor models and VAEs address these difficulties by imposing structure on the covariance matrix and provides low-rank approximation.

Through numerical experiments on historical stock returns, we demonstrate that VAEs provides the most accurate covariance matrix estimates compared to various benchmark methods. This also leads to a better minimum variance portfolio.

References

[1] B. M. Akesson and H. T. Toivonen. “A neural network model predictive controller”. In: Journal of Process Control 16 (2006), pp. 937–946.

[2] R. Almgren and N. Chriss. “Optimal Execution of Portfolio Transactions”. In: Journal of Risk 3.2 (2000), pp. 5–39.

[3] J. J. Angel. “Who gets price improvement on the NYSE”. In: Working Paper (1994).

[4] G. Y. Ban, N. E. Karoui, and A. E. B. Lim. “Machine learning and portfolio optimization”. In: Management Science 64.3 (2018), pp. 1136–1154.

[5] D. Bertsimas and A. W. Lo. “Optimal Control of Execution Costs”. In: Journal of Financial Markets 1 (1998), pp. 1–50.

[6] B. Biais, P. Hillion, and C. Spatt. “An empirical analysis of the limit order book and the order flow in the Paris Bourse”. In: The Journal of Finance 50.5 (1995), pp. 1655–1689.

[7] P. Carr, L. Wu, and Z. Zhang. “Using Machine Learning to Predict Realized Variance”. In: Working Paper (2019).

[8] J. W. Cho and E. Nelling. “The probability of limit-order execution”. In: Financial Analysts Journal 56(5) (2000), pp. 28–33.

[9] R. Coggins, A. Blazejewski, and M. Aitken. “Optimal Trade Execution of Equities in a Limit Order Market”. In: IEEE International Conference on Computational Intelligence for Financial Engineering 10.1109 (2003).

[10] R. Cont, S. Stoikov, and R. Talreja. “A stochastic model for order book dynamics”. In: Operations Research 58(3) (2010), pp. 549–563.

[11] V. Desai, V. Farias, and C. Moallemi. “Pathwise Optimization for Optimal Stopping Problems”. In: Management Science 58.12 (2012), pp. 2292–2308.

[12] M. Dixon, D. Klabjan, and J. H. Bang. “Classification-based financial markets prediction using deep neural networks”. In: Working Paper (2017).

[13] R. El-Yaniv et al. “Optimal Search and One-Way Trading Online Algorithms”. In: Algorithmica (2001), pp. 101–139.

[14] V. Francois-Lavet et al. “On Overfitting and Asymptotic Bias in Batch Reinforcement Learning with Partial Observability”. In: Journal of Artificial Intelligence Research 65 (2019).

[15] S. Gu, B. T. Kelly, and D. Xiu. “Autoencoder Asset Pricing Models”. In: Yale ICF Working Paper (2019).

[16] M. Haugh and L. Kogan. “Pricing American Options: A Duality Approach”. In: Operations Research 52.2 (2004).

[17] J. B. Heaton, N. G. Polson, and J. H. Witte. “Deep Learning in Finance”. In: Working Paper (2016).

[18] B. Hollifield, R. A. Miller, and P. Sandas. “Econometric analysis of limit-order executions”. In: The Review of Economic Studies 71(4) (2004), pp. 1027–1063.

[19] M. Kearns and S. Singh. “Bias-Variance Error Bounds for Temporal Difference Updates”. In:
COLT: Proceedings of the Thirteenth Annual Conference on Computational Learning Theory (2000), pp. 142–147.

[20] A. Kim, C. Shelton, and T. Poggio. “Modeling Stock Order Flows and Learning Market-Making from Data”. In: AI Memo (2002).

[21] O. Ledoit and M. Wolf. “Honey, I Shrunk the Sample Covariance Matrix”. In: UPF Economics and Business Working Paper 691 (2003).

[22] O. Ledoit and M. Wolf. “Improved Estimation of the Covaraince Matrix of Stock Returns With an Application to Porfolio Selection”. In: Journal of Empirical Finance 10.5 (2001), pp. 603–621.

[23] A. W. Lo, A. C. MacKinlay, and J. Zhang. “Econometric models of limit-order executions”. In: Journal of Financial Economics 65(1) (2002), pp. 31–71.

[24] F. Longstaff and E. Schwartz. “Valuing American Options by Simulation: A Simple Least-Squares Approach”. In: The Review of Financial Studies 14.1 (2001), pp. 113–147.

[25] H. Markowitz. “Portfolio Selection”. In: The Journal of Finance 7.1 (1952), pp. 77–91.

[26] C. C. Moallemi and K. Yuan. “A Model for Queue Position Valuation in a Limit Order Book”. In: Working Paper (2016).

[27] NASDAQ. “NASDAQ TotalView-ITCH 4.1”. In: (2010).

[28] Y. Nevmyvaka, Y. Feng, and M. Kearns. “Reinforcement Learning for Optimazed Trade Execution”. In: International Conference on Machine Learning (2006), pp. 673–680.

[29] A. Ntakaris et al. “Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods”. In: Working Paper (2018).

[30] A. Obizhaeva and J. Wang. “Optimal Trading Strategy and Supply/Demand Dynamics”. In: Journal of Financial Markets 16.1 (2013), pp. 1–32.

[31] B. Park and B. Van Roy. “Adaptive Execution: Exploration and Learning of Price Impact”. In: Operations Research 63.5 (2015), pp. 1058–1076.

[32] N. Passalis et al. “Temporal bag-of-features learning for predicting mid price movements using high frequency limit order book data”. In:
IEEE Transactions on Emerging Topics in Computational Intelligence (2018).

[33] M. A. Petersen and D. Fialkowski. “Posted versus effective spreads: Good prices or bad quotes”. In: Journal of Financial Economics 35.3 (1994), pp. 269–292.

[34] L.C.G. Rogers. “Monte Carlo valuation of American options”. In: Mathematical Finance (2003).

[35] J. Sirignano and R. Cont. “Universal Features of Price Formation in Financial Makrets: Persepectives From Deep Learning”. In: Quantitative Finance 19.9 (2019), pp. 1449–1459.

[36] J. A. Sirignano. “Deep Learning for Limit Order Books”. In: Quantitative Finance 19.4 (2019), pp. 549–570.

[37] Yoshiyuki Suimon et al. “Autoencoder-Based Three-Factor Model for the Yield Curve of Japanese Government Bonds and a Trading Strategy”. In:
Journal of Risk and Financial Management (2020).

[38] R. Sutton and A. Barto. Reinforcement Learning. ISBN 978-0-585-0244-5. MIT Press, 1998.

[39] M. Tipping and C. Bishop. “Probabilitic Principal Component Analysis”. In:
Journal of the Royal Statistical Society 61.3 (1999), pp. 611–622.

[40] I. M. Toke. “The order book as a queueing system: average depth and influence of the size of limit orders”. In: Quantitative Finance 15.5 (2013), pp. 795–808.

[41] D. T. Tran et al. “Temporal attention-augmented bilinear network for financial time-series data analysis”. In: IEEE transactions on neural networks and learning systems 30.5 (2019), pp. 1407–1418.

[42] D. T. Tran et al. “Tensor representation in high-frequency financial data for price change prediction”. In: IEEE Symposium Series (2017).

[43] A. Tsantekidis et al. “Forecasting stock prices from the limit order book using convolutional neural networks”. In: IEEE Business Informatics (CBI) (2017).

[44] J. Tsitsiklis and B. Van Roy. “Regression Methods for Pricing Complex American-Style Options”. In: IEEE Transactions on Neural Networks 12.4 (2001).

[45] H. van Hasselt, A. Guez, and D. Silver. “Deep Reinforcement Learning with Double Q-learning”. In:
AAAI’16: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (2016), pp. 2094–2100.

[46] R. Xiong, E. P. Nichols, and Y. Shen. “Deep Learning stock volatility with Google domestic trends”. In: Working Paper (2015).

[47] Z. Zhang, S. Zohren, and S. Roberts. “DeepLOB: Deep convolutional neural networks for limit order books”. In: Working Paper (2019).

Appendix A: Numerical Results for Chapter 3

A.1 Summary Statistics of Selected Stocks

Stock	Volume ($M)	Avg. Prices ($)	Price Vol.($)	Return Vol.	One Tick (%)	Spread
AAPL	94768.13	472.28	44.56	29%	0%	13.60
ADBE	3999.91	46.73	5.98	25%	81%	1.39
ADI	2441.46	46.57	2.16	20%	72%	3.26
ADP	3572.48	69.66	6.01	14%	59%	4.53
ADSK	2754.37	39.11	3.53	27%	70%	3.00
AMAT	3664.65	15.25	1.76	28%	99%	1.14
AMD	535.55	3.38	0.60	52%	98%	1.02
AMGN	9096.26	103.89	10.12	27%	18%	4.61
AMZN	17102.94	297.89	41.59	26%	0%	16.40
ATVI	2298.19	15.39	1.90	38%	98%	2.01
AVGO	2155.93	38.50	5.02	31%	61%	4.50
BAC	11935.64	13.44	1.31	23%	99%	1.01
BRK.B	3578.57	110.20	7.20	15%	11%	6.71
CHTR	1985.65	113.96	19.16	28%	3%	17.75
CMCSA	10030.18	43.23	3.40	20%	95%	1.27
COST	4996.78	111.86	7.00	14%	14%	4.58
CSCO	14958.94	22.69	1.83	34%	99%	1.02
CSX	2049.05	24.78	1.84	19%	96%	1.47
DIS	5904.70	62.83	5.71	18%	76%	1.81

EBAY	11696.10	53.45	1.88	25%	84%	1.46
F	4821.01	15.34	1.72	24%	98%	1.02
FB	32453.19	34.59	10.56	48%	93%	0.99
FISV	1481.28	91.68	11.36	53%	7%	8.31
GE	7809.10	23.99	1.62	17%	98%	1.04
GILD	11996.61	58.60	10.77	58%	66%	2.20
GS	7129.39	156.51	9.46	21%	2%	8.48
ILMN	1790.24	72.74	16.52	33%	7%	10.72
INTC	13742.28	23.05	1.34	20%	98%	0.99
INTU	3664.81	65.12	4.79	20%	47%	5.28
ISRG	4161.03	450.20	69.49	35%	0%	54.36
JNJ	10063.27	85.73	6.67	13%	75%	3.15
JPM	15719.47	51.85	3.35	19%	93%	1.71
LRCX	2413.90	47.00	4.62	25%	48%	3.19
MDLZ	6152.37	30.67	2.14	20%	97%	1.89
MELI	1224.31	109.33	16.36	37%	1%	27.96
MRK	7717.00	46.40	2.31	17%	93%	2.09
MSFT	27291.37	32.47	3.44	25%	98%	1.08
MU	9123.92	13.36	4.43	38%	98%	1.07
NFLX	15554.60	246.42	73.44	65%	0%	21.53
NVDA	2325.16	14.18	1.25	21%	98%	1.27
PEP	6836.76	80.35	4.14	14%	73%	3.14
QCOM	15814.58	66.36	3.38	18%	88%	2.37
SBUX	7015.92	67.38	9.18	19%	62%	1.46
T	8735.23	35.43	1.25	15%	97%	1.25
TXN	5857.73	37.61	3.27	18%	95%	1.42
UPS	4350.56	88.62	6.75	14%	42%	3.35

V	7143.93	180.35	16.65	21%	3%	15.47
VRTX	2983.46	68.01	13.45	70%	9%	10.04
VZ	7297.25	48.66	2.66	17%	92%	1.92
WFC	10620.15	40.17	3.27	16%	97%	1.11

Table A.1: Descriptive statistics for the selected 50 stocks over 2013. Average price and (annualized) volatility are calculated using daily closing price. Volume ($M) is the average daily trading volume in million dollars. One tick (%) is the percentage of time during trading hours that the spread is one tick. Spread is the time-averaged difference between best bid price and best ask price, in ticks.

A.2 State Variables

Category	**Features**
General	Time of day
Spread	Spread, Spread normalized by return volatility, Spread normalized by price volatility
Depth	Queue imbalance, Near depth, Far depth
Flow	Number of traders within the last second, Number of price changes within the last second
Intensity	Intensity measure for trades, price changes

Table A.2: Variables in States

- Queue Imbalance is defined as $\frac{\text{near depth} - \text{far depth}}{\text{near depth} + \text{far depth}}$. This can be calculated using depths at the top price levels and aggregated depth at the top 5 price levels.

- Intensity measure of any event is modeled as an exponentially decaying function with increments only at occurrences of such an event. Let S_t be the size of the trade (or price changes) at any given time t. $S_t = 0$ if there is no trade at time t. The intensity measure $X(t)$ can be modeled as

$$X(t + \Delta t) = X(t) \cdot \exp(-\Delta t / T) + S_{t+\Delta t}$$

A.3 Algorithm Performances

Stock	SL (Specific)	RL (Specific)	SL (Universal)	RL (Universal)	SL (Out)	RL (Out)
AAPL	37.36 (1.21)	44.8 (1.24)	38.57 (1.21)	43.23 (1.23)	38.9 (1.22)	44.4 (1.23)
ADBE	27.6 (0.80)	30.4 (0.81)	27.27 (0.81)	30.15 (0.81)	27.36 (0.80)	30.4 (0.81)
ADI	17.68 (1.09)	20.2 (1.10)	17.34 (1.08)	19.88 (1.09)	18.2 (1.08)	20 (1.09)
ADP	11.38 (1.07)	12.4 (1.08)	11.40 (1.07)	12.41 (1.09)	11.74 (1.08)	12.40 (1.10)
ADSK	30.58 (1.10)	34.20 (1.12)	29.57 (1.10)	33.67 (1.12)	29.48 (1.11)	33.40 (1.13)
AMAT	13.52 (0.71)	14.60 (0.72)	13.94 (0.71)	15.13 (0.72)	13.62 (0.72)	15.00 (0.73)
AMD	22.36 (0.72)	24.20 (0.73)	21.15 (0.72)	22.95 (0.73)	22.32 (0.73)	25.20 (0.74)
AMGN	37.98 (1.21)	44.60 (1.23)	38.89 (1.21)	45.67 (1.23)	41.96 (1.22)	46.80 (1.24)
AMZN	25.80 (1.32)	29.40 (1.35)	23.96 (1.32)	25.75 (1.35)	25.54 (1.33)	28.40 (1.35)
ATVI	18.58 (0.89)	20.60 (0.91)	21.35 (0.89)	22.68 (0.91)	22.58 (0.90)	22.60 (0.91)
AVGO	17.38 (1.05)	18.40 (1.07)	17.58 (1.05)	18.92 (1.07)	18.82 (1.06)	19.02 (1.08)
BAC	22.94 (0.71)	27.40 (0.72)	23.63 (0.71)	27.67 (0.72)	23.76 (0.72)	27.80 (0.73)
BRK.B	32.90 (1.25)	36.60 (1.28)	33.28 (1.25)	37.23 (1.28)	35.08 (1.26)	37.80 (1.28)
CHTR	12.74 (1.23)	16.20 (1.25)	12.72 (1.23)	16.16 (1.25)	13.82 (1.24)	17.40 (1.26)
CMCSA	17.16 (1.09)	21.00 (1.11)	16.64 (1.09)	20.17 (1.11)	17.70 (1.10)	21.00 (1.12)
COST	32.82 (1.31)	38.20 (1.34)	34.13 (1.31)	39.17 (1.34)	36.48 (1.32)	41.60 (1.34)
CSCO	15.16 (0.68)	17.60 (0.69)	14.99 (0.68)	16.93 (0.69)	15.60 (0.69)	17.40 (0.70)
CSX	14.74 (1.03)	16.20 (1.05)	15.09 (1.03)	16.49 (1.05)	15.22 (1.04)	17.40 (1.06)
DIS	18.44 (1.21)	21.40 (1.23)	19.54 (1.21)	22.89 (1.23)	20.62 (1.22)	23.40 (1.24)
EBAY	14.86 (1.19)	18.20 (1.22)	14.93 (1.19)	18.30 (1.22)	15.04 (1.20)	18.80 (1.22)
F	22.56 (0.89)	27.40 (0.91)	24.00 (0.89)	28.36 (0.91)	24.66 (0.90)	28.20 (0.91)
FB	15.68 (1.43)	16.20 (1.46)	15.91 (1.43)	16.46 (1.46)	16.04 (1.44)	16.60 (1.47)
FISV	21.36 (1.20)	24.20 (1.22)	21.76 (1.20)	24.65 (1.22)	22.96 (1.21)	24.40 (1.23)
GE	22.26 (0.68)	26.40 (0.69)	22.02 (0.68)	26.49 (0.69)	22.40 (0.69)	26.60 (0.70)
GILD	24.44 (0.90)	32.40 (0.92)	23.38 (0.90)	28.84 (0.92)	23.66 (0.91)	29.80 (0.92)
GS	28.38 (1.19)	34.40 (1.21)	26.80 (1.19)	31.82 (1.21)	27.24 (1.20)	32.20 (1.22)
ILMN	19.44 (1.22)	24.80 (1.24)	19.62 (1.22)	25.13 (1.24)	21.12 (1.23)	25.60 (1.25)
INTC	27.42 (0.75)	29.80 (0.77)	26.69 (0.75)	29.26 (0.77)	26.96 (0.76)	29.90 (0.77)
INTU	15.04 (1.11)	18.60 (1.13)	15.85 (1.11)	19.78 (1.13)	16.98 (1.12)	20.00 (1.14)
ISRG	15.92 (1.50)	19.00 (1.53)	17.39 (1.50)	21.05 (1.53)	19.50 (1.52)	21.80 (1.54)

JNJ	15.00 (1.09)	18.20 (1.11)	14.76 (1.09)	17.97 (1.11)	14.98 (1.10)	19.00 (1.12)
JPM	24.96 (0.80)	30.60 (0.82)	25.32 (0.80)	30.71 (0.82)	26.50 (0.81)	31.40 (0.82)
LRCX	17.04 (1.12)	20.20 (1.14)	16.81 (1.12)	21.08 (1.14)	17.36 (1.13)	21.80 (1.15)
MDLZ	11.92 (1.02)	14.20 (1.04)	12.66 (1.02)	14.97 (1.04)	12.74 (1.03)	14.40 (1.05)
MELI	13.90 (1.25)	15.20 (1.28)	14.42 (1.25)	15.72 (1.28)	15.14 (1.26)	17.00 (1.28)
MRK	27.74 (0.98)	34.20 (1.00)	28.33 (0.98)	34.96 (1.00)	29.38 (0.99)	36.40 (1.00)
MSFT	28.04 (0.81)	32.80 (0.83)	28.20 (0.81)	32.95 (0.83)	29.04 (0.82)	33.60 (0.83)
MU	36.30 (0.98)	36.60 (1.00)	34.86 (0.98)	35.87 (1.00)	35.06 (0.99)	36.40 (1.00)
NFLX	18.06 (1.39)	20.80 (1.42)	18.75 (1.39)	21.63 (1.42)	19.98 (1.40)	23.60 (1.42)
NVDA	16.64 (0.69)	18.00 (0.70)	16.96 (0.69)	18.48 (0.70)	16.82 (0.70)	19.00 (0.71)
PEP	13.78 (1.10)	18.40 (1.12)	13.52 (1.10)	18.12 (1.12)	14.42 (1.11)	18.80 (1.13)
QCOM	27.52 (0.77)	35.80 (0.78)	28.47 (0.77)	37.09 (0.78)	29.24 (0.77)	36.80 (0.79)
SBUX	38.26 (1.09)	41.40 (1.11)	37.05 (1.09)	39.84 (1.11)	37.94 (1.10)	39.60 (1.12)
T	18.06 (1.01)	20.40 (1.03)	17.10 (1.01)	19.65 (1.03)	17.92 (1.02)	20.60 (1.04)
TXN	11.22 (1.05)	12.40 (1.07)	11.66 (1.05)	12.72 (1.07)	12.16 (1.06)	13.40 (1.08)
UPS	15.54 (1.08)	16.20 (1.10)	16.38 (1.08)	17.37 (1.10)	18.84 (1.09)	19.20 (1.11)
V	24.46 (1.31)	29.60 (1.34)	25.15 (1.31)	29.47 (1.34)	25.90 (1.32)	29.80 (1.34)
VRTX	26.32 (1.19)	27.80 (1.21)	26.67 (1.19)	27.64 (1.21)	26.66 (1.20)	27.60 (1.22)
VZ	14.78 (0.93)	18.00 (0.95)	14.19 (0.93)	17.60 (0.95)	14.48 (0.94)	18.60 (0.96)
WFC	16.06 (1.05)	20.00 (1.07)	16.68 (1.05)	20.20 (1.07)	16.96 (1.06)	21.00 (1.08)
Avg.	21.40 (0.15)	24.82 (0.16)	21.55 (0.15)	24.85 (0.16)	22.34 (0.15)	25.47 (0.16)

Table A.3: These price gains are out-of-sample performances reported on the testing dataset. The numbers displayed are in percentage of the half-spread (% Half-Spread). The numbers in parenthesis are standard errors.

Appendix B: Proofs for Chapter 4

B.1 Lemma 4.4.1:

Let $D \in \mathbb{R}^{n\times n}$ be any diagonal matrix with non-negative entries, $L \in \mathbb{R}^{n\times k}$ is any $n \times k$ matrix, and I_k and I_n be the identity matrix of size $k \times k$ and $n \times n$ respectively.

Firstly, we will show the following corollaries:

Corollary 1:

$$\det\left[D(D + LL^\top)^{-1}\right] = \det\left[(I_k + L^\top D^{-1}L)^{-1}\right] \tag{B.1}$$

Proof. This is a direct application of Sylvester's determinant identity.

Equation (B.1) is equivalent to

$$\frac{\det(D)}{\det(D + LL^\top)} = \frac{1}{\det(I_k + L^\top D^{-1}L)}$$

Since $\det(D + LL^\top) = \det(D)\det(I_n + D^{-1}LL^\top)$, it suffices to show

$$\det(I_n + D^{-1}LL^\top) = \det(I_k + L^\top D^{-1}L)$$

This is true because of Sylvester's determinant identity.

□

Corollary 2:

Woodbury matrix identity gives us:

$$(D + LL^\top)^{-1} = D^{-1} - D^{-1}L(I_k + L^\top D^{-1}L)^{-1}L^\top D^{-1} \tag{B.2}$$

We will now use Bayes' rule and calculate to prove the Lemma 1

Proof.

$$
\begin{aligned}
p(z|x) &= \frac{p(x|z)p(z)}{p(x)} \\
&= \frac{\frac{1}{\sqrt{(2\pi)^n \det(D)}} \exp\left[-\frac{1}{2}(x - Lz)^\top D^{-1}(x - Lz)\right] \cdot \frac{1}{\sqrt{(2\pi)^k}} \exp\left[-\frac{1}{2}(z^\top z)\right]}{\frac{1}{\sqrt{(2\pi)^n \det(D+LL^\top)}} \exp\left[-\frac{1}{2}x^\top (D + LL^\top)^{-1}x\right]} \\
&= \frac{\sqrt{\det(D + LL^\top)}}{\sqrt{(2\pi)^k}\sqrt{\det(D)}} \cdot \exp\left\{-\frac{1}{2}(x - Lz)^\top D^{-1}(x - Lz) - \frac{1}{2}(z^\top z) + \frac{1}{2}x^\top (D + LL^\top)^{-1}x\right\} \\
&= \frac{1}{\sqrt{(2\pi)^k}\sqrt{\det\left[D(D + LL^\top)^{-1}\right]}} \cdot \exp\left\{-\frac{1}{2}z^\top (I_k + L^\top D^{-1}L)z + x^\top D^{-1}Lz\right\} \\
&\quad \cdot \exp\left[-\frac{1}{2}x^\top D^{-1}x + \frac{1}{2}x^\top (D + LL^\top)^{-1}x\right]
\end{aligned}
$$

Now make use of proposition 1 and let's also denote $S = (I_k + L^\top D^{-1}L)^{-1}$. This leads to the following.

$$
\begin{aligned}
p(z|x) &= \frac{1}{\sqrt{(2\pi)^k \det(S)}} \cdot \exp\left\{-\frac{1}{2}z^\top S^{-1}z + x^\top D^{-1}Lz\right\} \cdot \exp\left[-\frac{1}{2}x^\top D^{-1}x + \frac{1}{2}x^\top (D + LL^\top)^{-1}x\right] \\
&= \frac{1}{\sqrt{(2\pi)^k \det(S)}} \cdot \exp\left[-\frac{1}{2}(z - SL^\top D^{-1}x)^\top S^{-1}(z - SL^\top D^{-1}x)\right] \cdot \exp\left[\frac{1}{2}(x^\top D^{-1}LSL^\top D^{-1}x)\right] \\
&\quad \cdot \exp\left[-\frac{1}{2}x^\top D^{-1}x + \frac{1}{2}x^\top (D + LL^\top)^{-1}x\right]
\end{aligned}
$$

Using (B.2), the last two exponential terms above can be canceled to 1. Specifically,

$$\exp\left[\frac{1}{2}(x^\top D^{-1}LSL^\top D^{-1}x)\right]\cdot\exp\left[-\frac{1}{2}x^\top D^{-1}x+\frac{1}{2}x^\top(D+LL^\top)^{-1}x\right]$$

$$=\exp\left\{\frac{1}{2}\left[x^\top\left(D^{-1}LSL^\top D^{-1}-D^{-1}+(D+LL^\top)^{-1}\right)x\right]\right\}$$

$$=\exp(0)=1$$

This completes the calculation of $p(z|x)$ as below.

$$p(z|x)=\frac{1}{\sqrt{(2\pi)^k\det(S)}}\exp\left[-\frac{1}{2}(z-SL^\top D^{-1}x)^\top S^{-1}(z-SL^\top D^{-1}x)\right]$$

This is just the pdf of $N\left(SL^\top D^{-1}x,S\right)$, and thus $z|x\sim N\left(SL^\top D^{-1}x,S\right)$ □

In the lemma, $D=\sigma^2 I$. Plugging in to the above results gives us the following

$$z|x\sim N(\sigma^{-2}SL^\top x,S).$$

B.2 Theorem 4.4.3

Proof. The training problem of the VAE represented in (4.27) can be formulated as follow.

$$\begin{aligned}\max_{L,\sigma,\phi,\Sigma}\quad & \mathbb{E}_{z\sim Q}\left[\log P(X|z)\right]-\mathcal{D}_{KL}\left[Q(z|X)||P(z)\right]\\ \text{s.t.}\quad & P_{L,\sigma}(X|z)=N\left(Lz,\sigma^2 I\right)\\ & Q_{\phi,\Sigma}(z|X)=N\left(g_\phi(X),\Sigma\right)\\ & \Sigma\succeq 0\end{aligned}\tag{B.3}$$

Note that the objective function is the ELBO. The equivalence in (4.17) allows us to reformulate

the above optimization problem as the following.

$$\begin{aligned} \max_{L,\sigma,\phi,\Sigma} \quad & \log P(X) - \mathcal{D}_{KL}\left[Q(z|X)||P(z|X)\right] \\ \text{s.t.} \quad & P_{L,\sigma}(X|z) = N\left(Lz, \sigma^2 I\right) \\ & Q_{\phi,\Sigma}(z|X) = N\left(g_\phi(X), \Sigma\right) \\ & \Sigma \succeq 0 \end{aligned} \tag{B.4}$$

Define the objective function as

$$\mathcal{L}(L,\sigma,\phi,\Sigma) = \log P_{L,\sigma}(X) - \mathcal{D}_{KL}\left[Q_{\phi,\Sigma}(z|X)||P_{L,\sigma}(z|X)\right]. \tag{B.5}$$

Notice that since $P(X) = \int P_{L,\sigma}(X|z)P(z)dz$, $P(X)$ only depends on the generative parameters (L,σ). Similarly, since

$$P(z|X) = \frac{P(X|z)P(z)}{P(X)}$$

, $P(z|X)$ also only depends on generative parameters (L,σ) as well. On the other hand, the approximate posterior $Q_{\phi,\Sigma}(z|X)$ is characterized by the variational parameters (ϕ,Σ).

We want to show that $(L^*,\sigma^*,g^*,\Sigma^*)$ defined in Theorem 4.4.3 is a set of optimal solution to the above optimization problems. In order to achieve this, we show that any other set of of parameters (L,σ,g,Σ) will result in a objective value that is no higher than that of $(L^*,\sigma^*,g^*,\Sigma^*)$. In other words,

$$\mathcal{L}(L,\sigma,\phi,\Sigma) \leq \mathcal{L}(L^*,\sigma^*,g^*,\Sigma^*) \qquad \forall (L,\sigma,g,\Sigma).$$

Firstly, from Lemma 4.4.1, for any set of generative parameter (L,σ), the true posterior is given by

$$P(z|x) = N(\sigma^{-2}SL^\top x, S),$$

where $S = (I_k + \sigma^{-2}L^\top L)^{-1}$. The approximate posterior that matches the true posterior will yield a KL-divergence value of 0. Therefore, for any set of generative parameter (L,σ), define the

corresponding variational parameters $(g_\phi^{L,\sigma}, \Sigma^{L,\sigma}) = (\sigma^{-2} S L^\top x, S)$, then the KL-divergence of the approximate posterior and the true posterior is 0.

Because KL-divergence is always non-negative, any other set of variational parameter will result in a non-negative KL-divergence. That is,

$$\mathcal{D}_{KL}\left[Q(z|X)||P(z|X)\right] \geq 0. \tag{B.6}$$

Secondly, according to Tipping and Bishop 1999, the PCA solution maximizes the log-likelihood. In other words, let (L^*, σ^*) be the PCA estimates, then for any generative parameter (L, σ), the following is true

$$\log P_{L,\sigma}(X) \leq \log P_{L^*,\sigma^*}(X) \tag{B.7}$$

Combining (B.6) and (B.7), we conclude that $(L^*, \sigma^*, g^*, \Sigma^*)$ defined in Theorem 4.4.3 is a set of optimal solution to the optimization problem in (B.4). This completes the proof. □