Division of Engineering Science
UNIVERSITY OF TORONTO

An Application of Deep Reinforcement Learning for Order Execution

with a supporting analysis of

Optimal Liquidation under a Transient Price Impact Model

Matthew B. Reiter
Division of Engineering Science, University of Toronto

Supervised by Dr. Y. Lawryshyn
Associate Professor, Faculty of Applied Science and Engineering, University of Toronto
Director, Centre for Management of Technology and Entrepreneurship

With guidance from Dr. M. Fukasawa
Professor, Graduate School of Engineering Science, Osaka University

A thesis submitted in requirement for the degree of

Bachelor of Applied Science

Toronto 2020

Centre for Management
of Technology and
Entrepreneurship

Dedicated to Sarah.

Acknowledgements

To the following, I offer my sincere gratitude:

Dr. Y. Lawryshyn, for continually supporting my ambitions and committing your time to ensure the success of this project.

Dr. M. Fukasawa, for introducing me to the subject of Order Execution and graciously hosting me in your laboratory group.

Mr. C. Geoffrey, for providing assistance and guidance when I needed to navigate the RIT environment.

David S, for outlining a challenging problem and offering insights to overcome the many challenges.

Zarir G, for being a steadfast mentor and friend.

My family, for your endless support. **Lauren**, I am ever thankful for your help with editing.

Abstract

A trader who needs to sell or purchase a large quantity of shares has to decide on an execution strategy to follow. A rational trader would strive to realize the optimal trading policy which returns the largest profits whilst striking a balance between price impact risk and price uncertainty risk. To make informed decisions, the trader needs to understand and take into account the complicated market factors that drive the price of securities on a real exchange. The study of Order Execution formalizes the models and optimization objectives which arise from profit maximizing trading incentives. Owing to the complexity however, traditional approaches for Order Execution are plagued by intractability under generalized conditions. An alternative method is to leverage the capabilities of Deep Reinforcement Learning, and train an autonomous agent to trade optimally under complex market dynamics, all with reduced modelling assumptions.

The culmination of this thesis develops an Asynchronous Advantage Actor-Critic model that trains an agent to perform Volume-Weighted Average Price execution. The agent achieves modest profits, and further shows evidence of tracking to the price target. The application of Deep Reinforcement Learning to Order Execution is fortified by two supplementary topics; the first relating to Transfer Learning and the second, covering a models-dependent approach to Order Execution. The treatment of Transfer Learning demonstrates that a reference Deep Reinforcement Learning model can be trained more efficiently under a Teacher/Student learning framework. Specifically, the Student model is shown to learn at an accelerated rate when given parameter preinitialization from the Teacher, in addition to receiving both an advice and observation stream in the decisioning network. The models-dependent approach to Order Execution is achieved under a transient price impact model, and the solution gives insight into how economic factors – market depth, resilience, tightness and bias – influence optimal trading decisions. The approach to solve the optimal liquidation objective on the transient price impact model is first done through Quadratic Programming and then through the Calculus of Variations, demonstrating a simple, yet robust method.

Contents

List of Figures

Executive Summary

In studying an application of Deep Reinforcement Learning for Order Execution, we address three topics of work and arrive at recommendations for each.

The first topic in this thesis deals with Teacher/Student learning models. Under the Teacher/Student learning framework, a Teacher model is first trained to perform a comparable, but different, task to the one that a Student model must learn. Two schemes are devised – the *Advice-Driven* decisioning approach and the *Advice & State-Driven* decisioning approach – which outline how the Student model handles advice and transferred knowledge from the Teacher. The intended result is to capture a reduction in the time that it takes to train the Student model.

▶ **Recommendation 1:** For the Teacher/Student learning problem, it is effective to initialize the Student model with the network parameters from the Teacher model. Additionally, a significant reduction in training time is observed under the *Advice & State-Driven* decisioning approach – that is, when the Student model receives both Teacher-recommended advice and an observation on the state as feature inputs.

The second topic studies Optimal Liquidation under a transient price impact model. The price dynamics lead to the derivation of a mean-variance optimization problem that maximizes a trader's risk-adjusted liquidation wealth. The motivation for studying the transient price impact model is to arrive at an intuitive understanding of how various market factors – depth, resilience, tightness and bias – influence optimal trading decisions.

▶ **Recommendation 2:** The transient price impact model can be solved in discrete-time, and further studied with continuous-time representation, to characterize a richly diverse set of optimal trading policies under different agent- and market-specific factors.

The last topic introduces Volume-Weighted Average Price (VWAP) execution within the context of Deep Reinforcement Learning and specifically, an Asynchronous Advantage Actor-Critic (A3C) model. The methodology presents three training environments that are conducive to the model and outlines the training scheme that performs Order Execution with VWAP as a benchmark execution price.

▶ **Recommendation 3:** The A3C model is able to modestly track VWAP, but underperformed by three basis-points on average during testing. The limitations of the modelling scheme introduce an opportunity to consider an enhanced reward signal, whereby an *Advice & State-Driven* learning model is a prime candidate for supplemental training.

Chapter 1

Introduction

The subject of Deep Reinforcement Learning continues to receive widespread attention from academics and practitioners alike. With the rapid pace of development, the dedication to the subject lies in the prospect of leveraging Artificial Intelligence to capture operational efficiencies and improve the scalability of data-driven systems. As technology continues to improve in terms of processing capabilities, the applications for Deep Reinforcement Learning will become more common and drive new business solutions.

This thesis deals with an application of Deep Reinforcement Learning as it pertains to *Order Execution*. Order Execution describes a general class of problems where a trader is tasked with purchasing or liquidating a target number of shares while minimizing transaction costs and uncertainties in the execution price. Technicalities in the execution of trades result from the microstructure of financial markets, distinctly due to the dynamics that govern the interactions between intermediaries. Transaction costs pose a threat to profit-generating trading strategies, and in response financial institutions dedicate computational resources and research focus to mitigate adverse market impacts. Academics studying analytical solutions to Optimal Order Execution problems borrow from topics in Stochastic Control to model the intricate relationships that dictate market factors such as liquidity, price evolution and resiliency. Alternatively, an approach leveraging Deep Reinforcement Learning forgoes the complicated models upfront and banks on the concept of embedding the optimal-handling of these market dynamics in the decisioning of an autonomous agent. In devising an application, this work studies the feasibility of Deep Reinforcement Learning as it applies to Order Execution.

Additionally, this work studies the intrinsic value of a trained model in terms of what domain-specific information can be transferred to a relatable setting. This attempt at Teacher/Student learning, a branching topic in the subject of Transfer Learning, is applied to several Atari 2600 arcade games prior to suggesting an application within the context of Order Execution. The subject of Transfer Learning similarly carries large appeal for the sake of efficient training and has been considered by Andrew Ng, chief scientist at Baidu and professor at Stanford, to be “the next driver of commercialized machine learning success.”[1]

1.1 Context and Motivation

This project has been completed with support from the Centre for Management of Technology and Entrepreneurship (CMTE) at the University of Toronto. The primary sponsor for this research is a prominent Canadian financial institution, one that is providing Deep Reinforcement Learning-based solutions to enhance a prime brokerage service offering. Henceforth, any reference made to the sponsoring financial institution shall be done by addressing “the Bank”.

Work for this thesis has been completed over an academic period beginning in May 2019and ending in April 2020, with planning and preparation dating back to February 2019. Serving as a required credit for the completion of the BASc degree in Engineering Science at the University of Toronto, material for this work has been delivered in accordance to the requirements and guidelines defined in ESC499Y1. Resources, both computational and in counsel, have been made available without barrier by the CMTE, the Bank and the Division of Engineering Science, and the Faculty of Applied Science and Engineering at the University of Toronto.

1.2 Objectives

The objective of this thesis is to efficiently solve an Order Execution objective through a Deep Reinforcement Learning approach. In doing so, a Teacher/Student learning framework is studied in supplement, to demonstrate how a reference model can be trained more efficiently and in less time. We further support the Order Execution analysis with a review of Optimal Liquidation under a transient price impact model. Given in detail below are the three primary objectives.

1. The first objective is to demonstrate that a Teacher/Student learning framework can train a reference model, referred to as a *Student*, at an accelerated rate when compared to a baseline model. An Asynchronous Advantage Actor-Critic algorithm was utilized and training occurred on three emulated Atari 2600 arcade games: *Breakout*, *Beamrider* and *Space Invaders*. To demonstrate the application of Teacher/Student learning, the Teacher models were trained for each environment under a modified reward signal. Then, two variants of Student learners were defined, both accepting advice from the Teacher and one incorporating a state-observation stream that supports the preinitialization of network parameters from the Teacher.

[1]NIPS 2016 tutorial entitled “Nuts and bolts of building AI applications using Deep Learning”

By demonstrating the effectiveness of Teacher/Student learning, the main result secured a computationally efficient training scheme that is suggested for an extension to the Deep Reinforcement Learning model for Order Execution. Work for this objective is covered in chapter 2 and the discussions provide additional commentary about reward signal engineering and hyperparameter tuning.

2. The second objective motivates the Order Execution task by studying Optimal Liquidation under a transient price impact model. As a models-dependent approach to Order Execution, a variant of the block shaped limit order book is studied with coupled price dynamics for the bid- and ask-price processes that account for market liquidity and resiliency factors. The optimization problem is formulated in continuous-time for a trader with constant absolute risk-aversion and correspondingly, the optimization problem assumes the common form of a mean-variance maximization for liquidation wealth. The second objective is detailed in chapter 3.

   Original contributions include: taking a discretization to numerically solve the system as a Quadratic Program, taking a limit of the discrete model under simplifying conditions to verify results from [4], and formulating a continuous-time solution with dynamics similar to the results from [59]. The results outline a numerically feasible routine for the optimal scheduling of a share liquidation program. Many of the insights then carry directly forward into the final objective with this work.

3. The final objective is the application of Deep Reinforcement Learning for Order Execution. The supporting mechanism is drawn from an Asynchronous Advantage Actor-Critic model. To facilitate the training of the model, three distinct trading environments were developed that extend the classes provided by *OpenAI*[2]. The first environment derives from the block shaped limit order book studied in chapter 3, the second environment utilizes historical data from the Nasdaq stock exchange, and the third environment is based on an interactive simulated trading platform maintained by the Rotman School of Management at the University of Toronto. The environments have also been utilized by a colleague for a different application. All work pertaining to this last objective is found in chapter 4.

   The methodology introduces the Deep Reinforcement Learning model. Within the context of Order Execution, the agent is trained to realize an execution price that matches or bests the Volume-Weighted Average Price (VWAP) over the execution horizon. Defining an appropriate reward signal proves critical to the agent learning an effective liquidation strategy, and a Time-Weighted Average Price (TWAP) tracking incentive is used as a baseline for the reward signal. Connections to the material studied in chapter 2 cover the adaptation of a Teacher/Student scenario for Order Execution and a recommendation is provided as to how such a model may be implemented.

[2]https://gym.openai.com/

Chapter 2

Deep Reinforcement Learning and Teacher/Student Learning

The content in this chapter centres around Teacher/Student learning models. The fundamental background and related literature is introduced in section 2.1, providing a high-level summary of fundamental concepts. Notational conventions will also be introduced this section, thereon defining the standard that carries forward throughout this body of work.

The methodology portion is detailed in section 2.2, which presents the Deep Reinforcement Learning structure and covers the training schemes for the Teacher and Student models. The models in this chapter are trained on three Atari 2600 arcade games, and the learning curves are given in section 2.3 with an accompanying discussion. The chapter concludes with section 2.4, on a note recognizing areas of improvement and suggesting future adaptations to the Teacher/Student learning framework.

2.1 Background

2.1.1 Deep Learning

Deep Learning describes a class of models which attempt to *learn* a functional mapping between an input and output space. Deep Learning approaches fall under the larger domain of Machine Learning, and as a result there are several inherited concepts, including the very nature of what it means to *learn*. Within the context of Machine Learning, a well-posed learning problem may be characterized by the following quoted-definition:

Definition 2.1.1 *A computer program is said to learn from experience $\mathcal{E}$ with respect to some class of tasks $\mathcal{T}$ and performance measure $\mathcal{P}$, if its performance at tasks in $\mathcal{T}$, as measured by $\mathcal{P}$, improves with experience $\mathcal{E}$. [86]*

Definition 2.1.1 offers insight and even hints towards a modular approach when it comes to designing and implementing Machine Learning algorithms. While there is no single routine for a design approach, there are generally three steps to follow and be mindful of:

1. The design approach typically begins by selecting a model which is believed to capture a systematic relationship between features, which are the inputs to the model, and targets, which are the intended outputs of the model. Machine Learning problems are wide-ranging, in part, due to the flexibility in choosing different models. When subjugating Deep Learning problems, the model structure is typically a variant of a Neural Network structure.

2. When a model structure is chosen, the natural progression is to quantify the deficiency of the model, or put in other words, how poorly the model is able to capture the relationship between features and targets. The measure of fit is referred to as a loss function and also offers a significant degree of freedom when outlining a Machine Learning algorithm.

3. The last guideline is to follow an optimization routine which attempts to improve the model's performance by minimizing the loss function. An optimization scheme may be analytically derived; however, a numerical solution is commonly adopted due to the complexity of the model and constraint-space.

In relating back to Definition 2.1.1: the "experience $\mathcal{E}$" refers to the data under which the model is trained to; the "class of tasks $\mathcal{T}$" and "the performance measure $\mathcal{P}$" describe the model's objective and loss function respectively; and the model "improving with experience $\mathcal{E}$" connotes the optimization to minimize the loss. Certainly there are other interpretations, but as a starting point Definition 2.1.1 will be helpful to consider as we advance discussions on Deep Learning.

Due to the data-intensive nature of designing a Deep Learning model, specifically in regards to the dimensionality and scope of the input space, the computational expense is often large. Navigating this challenge continues be simplified owing to recent advances in technological systems and processing capabilities, proving the effectiveness of Deep Learning in areas such as Image Classification [3], Speech Recognition [2] and as we will soon cover, Reinforcement Learning. Deep Learning models, specifically Neural Networks, demonstrate promise due to their vast expressiveness, deriving from an ability to universally approximate functions to an arbitrary level of precision [6][5][24][60].

2.1.1.1 Feedforward Neural Networks

Feedforward Neural Networks (FNN's) remain one of the central tenants to a Deep Learning model. Inspired by the neural pathways in the animal brain, the functional units in an FNN are referred to as perceptrons which are arranged sequentially in a series of fully-connected layers. The input layer is directly exposed to the features and the internal "hidden" layers are fully connected to the perceptrons of the previous layer. The computation performed by each perceptron is an affine transformation of all the sources of input.

Figure 2.1.1 presents a simple example of an FNN with two feature inputs, and a single hidden layer with three perceptrons providing output to a single quantity. In further reference to the structure of the FNN – and unless otherwise specified – we will refer to a collection of perceptrons by their layer, we will treat inputs as corresponding to all perceptrons in the previous layer, and we will take outputs as all perceptrons in the following layer.

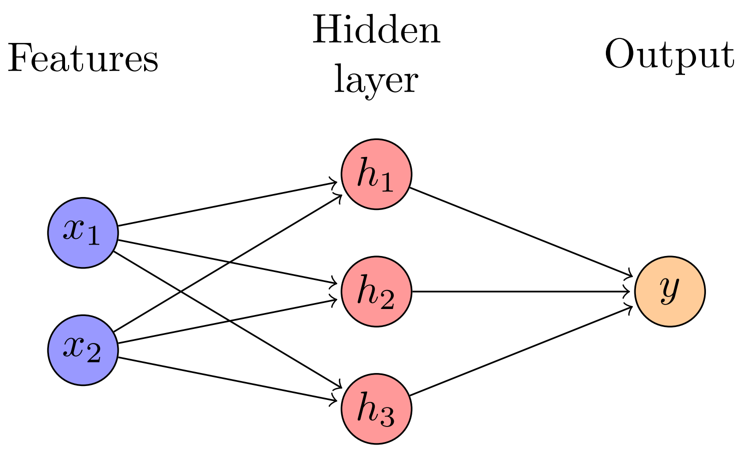
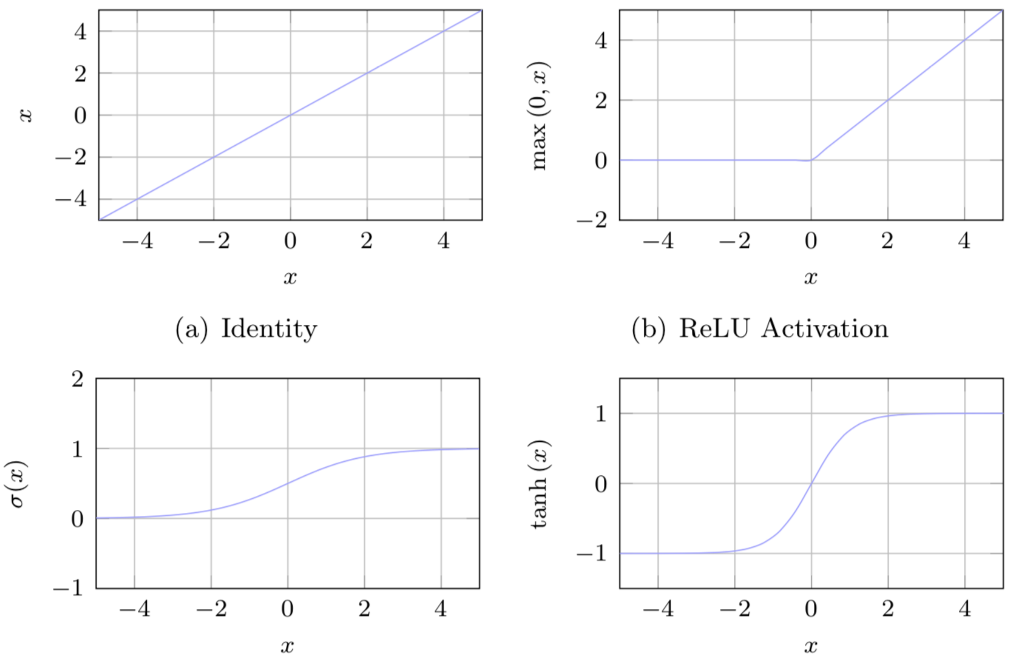


Figure 2.1.1: An example of a Feedforward Neural Network with a single output given from two input features and one hidden layer.

Note that the structure of an FNN is analogous to a directed, acyclic graph. Correspondingly, the connecting edges between layers are prescribed weights $w_{ji}^{(k)} \in \mathbb{R}$. For clarity, $w_{ji}^{(k)}$ is the weight for the edge connecting perceptron j in layer k to the input perceptron i. If the hidden layer k contains K perceptrons, then we express the vector $\mathbf{h}^{(k)} = (h_j^{(k)}) \in \mathbb{R}^K$ and if the input layer contains I inputs, then we express the vector $\mathbf{h}^{(k-1)} = (h_i^{(k-1)}) \in \mathbb{R}^I$. For this vectorized setup, the accompanying weights are expressed in a matrix $\mathbf{W}^{(k)} = (w_{ji}^{(k)}) \in \mathbb{R}^{K \times I}$. Then for Figure 2.1.1, the affine transformation for the hidden layer may be written

$$\mathbf{h}^{(1)} = \phi^{(1)}\Big(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\Big)$$

With bias term $\mathbf{b}^{(1)} = (b_j^{(1)}) \in \mathbb{R}^3$ and activation function $\phi^{(1)} : \mathbb{R} \to \mathbb{R}$. The bias offset may be viewed as another input dimension and the activation function, in a general sense, provides an additional feature-mapping to control the range of the output. Several common activation functions are presented in Figure 2.1.2, all of which are utilized in the networks constructed in section 2.2.



(a) Identity (b) ReLU Activation (c) Sigmoid Activation (d) Hyperbolic Tangent Activation

Figure 2.1.2: Common activation functions used in Neural Network structures.

The weights and biases for all layers in an FNN are the parameters which need to be optimized to tune the performance of the model. In doing so, we introduce the concept of *Backpropagation*.

2.1.1.2 Backpropagation

Recall that a loss function is the performance metric for a Machine Learning model which quantifies how poorly the model performed at its task. In a *Supervised Learning* setting, features are accompanied with corresponding labels, or “true” output values, in which case the loss function may be taken, for example, as the mean-squared error. However in a different context, as we will soon discuss within the realm of Reinforcement Learning, the loss function must be crafted against a different and more subtle heuristic. For this reason, we proceed by allowing for a general definition of the loss function $\mathcal{L} : (y; \mathcal{H}) \to \mathbb{R}$, which scores the model’s output y against the heuristic $\mathcal{H}$.

Once again using Figure 2.1.1 as the basis for an example, the output y is written:

$$\begin{aligned} y &= \phi^{(2)}\Big(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\Big) \\ &= \phi^{(2)}\Big(\mathbf{W}^{(2)}\phi^{(1)}\Big(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\Big) + \mathbf{b}^{(2)}\Big) \end{aligned}$$

This reveals how the network's output acts a functional composition of the hidden layers. Then, within the scope of minimizing the loss function, gradients with respect to the network's parameters may be computed using the chain rule. This however poses a computational challenge since as the network grows in complexity, the number of gradients to compute becomes large. This computational hurdle is what backpropagation [23] addresses. The backpropagation algorithm offers an efficient way to calculate the gradients in a network and follows quite naturally from an understanding of the chain rule. The algorithm begins after first completing a forward pass through the network, which serves the purpose of calculating the loss through the layers. For the network shown in Figure 2.1.1, the forward pass and the accompanying backpropagation is presented below:

Forward Pass:

$$\begin{aligned} \mathbf{h}^{(1)} &= \phi^{(1)}\Big(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\Big) \\ y &= \phi^{(2)}\Big(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\Big) \end{aligned}$$

Backpropagation:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}} &= \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial \mathbf{W}^{(2)}} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(2)}} &= \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial \mathbf{b}^{(2)}} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(1)}} &= \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial \mathbf{h}^{(1)}} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(1)}}\frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}^{(1)}} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(1)}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(1)}}\frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{b}^{(1)}} \end{aligned}$$

Following the backpropagation approach, gradients are first computed at the output level then computed for each feeding layer there prior, thus propagating errors back to the start of the network. The optimization of the parameters still relies on an update rule and guideline, of which the standard routine is gradient descent [75]. The update rule for gradient descent with respect to parameter $\mathbf{W}^{(k)}$ is presented immediately below:

$$\mathbf{W}^{(k)} \leftarrow \mathbf{W}^{(k)} - \alpha\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(k)}}$$

Gradient descent is an iterative approach with the learning rate α being a parameter that controls the rate at which an update proceeds along the direction indicated by the gradient. The learning rate is treated as a hyperparamter, and the tuning of which poises a tradeoff between computational overhead and numerical stability. Gradient descent is considered a baseline because it captures the theme of numerical optimization, however, computing the gradient for the update requires batch training over the entirety of training dataset. Even with backpropagation, a large training dataset will lead to computational inefficiencies for batch training approaches that ultimately culminate to an infeasible training scheme.

The practical solution is mini-batch training, and forms the basis for a common adaptation of *Stochastic Gradient Descent*. Stochastic Gradient Descent with mini-batch training involves computing gradients over a smaller, randomly sampled subset of the training data. The justification for this sampling approach rests from the fact that the stochastic gradient will be an unbiased estimator for the batch gradient provided that batch-sampling is conducted uniformly at random.

In addition to Stochastic Gradient Descent, other optimization variants include RMSprop [85] and Adam [51], which are both designed around the concept of adaptively configuring learning rates. Adaptive models have become popular as of late due to empirical results that demonstrate efficient computation and near-guaranteed outperformance of the benchmark models which they approximate [20]. Relevant to this body of work is Adam – the optimizer of choice for the models presented in section 2.2 – which works to configure learning rates on a per parameter basis.

Up until this point, discussions on Neural Networks and the optimization of their parameters have been limited in scope, and in example, to FNNs. However, there remains several other variants of Neural Network structures commonly used in Deep Learning methods, where the techniques of backpropagation and numerical optimization certainly still apply. In turn, the upcoming topic for background discussions are *Long Short-Term Memory Cells*.

2.1.1.3 Long Short-Term Memory Cells

FNN's are acylic, and as a result, information flows through the network in a single direction. When directed cycles are introduced, the resulting network is referred to as a *Recurrent Neural Network* (RNN) [93]. RNN's are well suited to address tasks which require a component of sequence recognition, such as Natural Language Processing [96], or has an input with temporal dependence, as in time series forecasting [41]. The structure of an RNN may be "unravelled", as shown in Figure 2.1.3, to visualize the flow of information through the network.

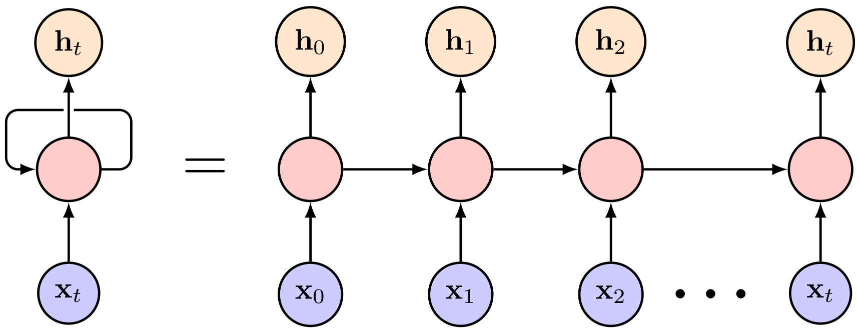


Figure 2.1.3: An unravelled Recurrent Neural Network.

Resulting from the sequential dependence, RNN structures carry a memory of the input which allows for previous information about the cell state to persist. However, when previous information becomes stale, the internal structure of a *Long Short-Term Memory Cell* (LSTM) can facilitate the network "forgetting" the stale information and focus inference from more recent input data [42]. The structure of an LSTM is shown in Figure 2.1.4. Each cell receives input data $\mathbf{x}_t$ in addition to the network's previous state $\mathbf{c}_{t-1}$ and output state $\mathbf{h}_{t-1}$.

There are three gates in the structure of an LSTM. The first is responsible for dictating which information is forgotten from the network, taking $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ as inputs. The output $\mathbf{f}_t$, receiving a sigmoid activation, passes through to a hadamard product on $\mathbf{c}_t$. The next gate consists of two operations with the purpose of proposing a candidate state $\hat{\mathbf{c}}_t$ which is to be amended, through addition, to the previous modified cell state value $\mathbf{f}_t \circ \mathbf{c}_{t-1}$. The last gated operation involves formulating the output $\mathbf{h}_t$ by taking another hadamard product on the now final cell state $\mathbf{c}_t$, with sigmoid-activated weighted selections of $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$. The described operations are summarized below:

$$\begin{aligned}
\mathbf{f}_t &= \sigma\left(\mathbf{W}_h^{(f)}\mathbf{h}_{t-1} + \mathbf{W}_x^{(f)}\mathbf{x}_t + \mathbf{b}^{(f)}\right)\\
\mathbf{i}_t &= \sigma\left(\mathbf{W}_h^{(i)}\mathbf{h}_{t-1} + \mathbf{W}_x^{(i)}\mathbf{x}_t + \mathbf{b}^{(i)}\right)\\
\hat{\mathbf{c}}_t &= \tanh\left(\mathbf{W}_h^{(\hat{c})}\mathbf{h}_{t-1} + \mathbf{W}_x^{(\hat{c})}\mathbf{x}_t + \mathbf{b}^{(\hat{c})}\right)\\
\mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \hat{\mathbf{c}}_t\\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_h^{(o)}\mathbf{h}_{t-1} + \mathbf{W}_x^{(o)}\mathbf{x}_t + \mathbf{b}^{(o)}\right)\\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh\left(\mathbf{c}_t\right)
\end{aligned}$$

LSTMs are not the only type of gated RNN, with various other structures generalized as Gated Recurrent Units (GRUs) [19]. Additionally, other variants incorporate information bidirectionally [35]; however, for the material introduced in section 2.2, the LSTMs as described above will be utilized.

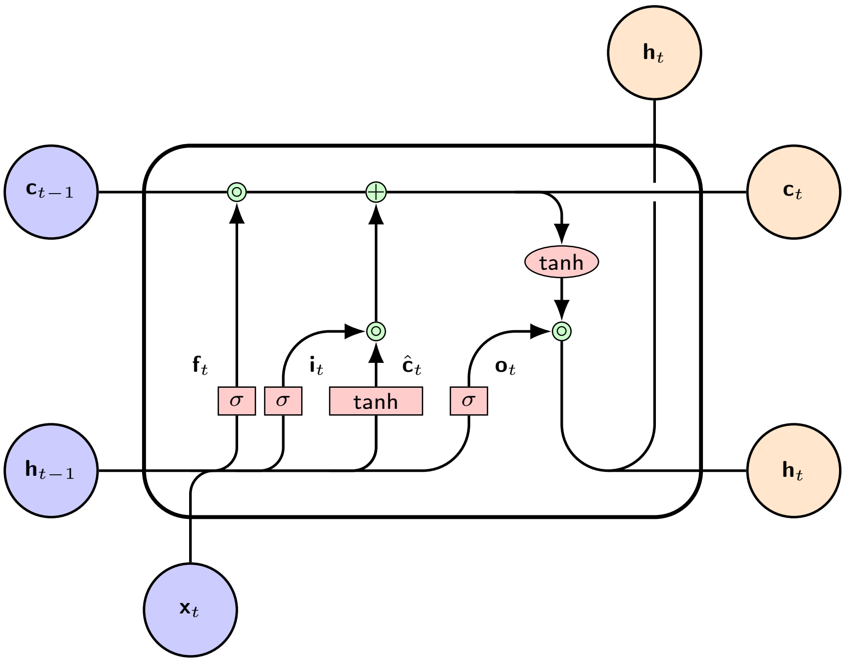


Figure 2.1.4: A diagram of an LSTM cell.

The final Deep Learning construct that will be covered in this section is the *Convolutional Neural Network*.

2.1.1.4 Convolutional Neural Networks

By performing a convolution operation on input data, a Convolutional Neural Network (CNN) is able to infer proximal and spacial relationships in the feature space. An example of a two-dimensional, one-stride convolution operation with a (3×3) kernel is provided in Figure 2.1.5. From this view, the convolution operation works by *striding* over subsets of the input and applying a weighted-sum to produce an output. The weights given by the kernel are the parameters that the network is trained to optimize. One notable efficiency with convolutional networks is that weights are shared – given at each layer on a kernel-basis – and thus keeps the number of parameters bounded by the dimension of the input, as realized in a degenerate case with a unit kernel.

CNNs are typically constructed by sequencing together several convolutional layers. Other common techniques such as max-pooling may also be applied, however, researchers in [84] demonstrate this is not required to achieve good performance. CNNs have become a staple in tasks pertaining to image classification, owing to the sweeping success of [3], but various methodologies and justifications exist to apply CNNs on non-image input data [80] [98], which becomes relevant in chapter 4 when we study Order Execution models.

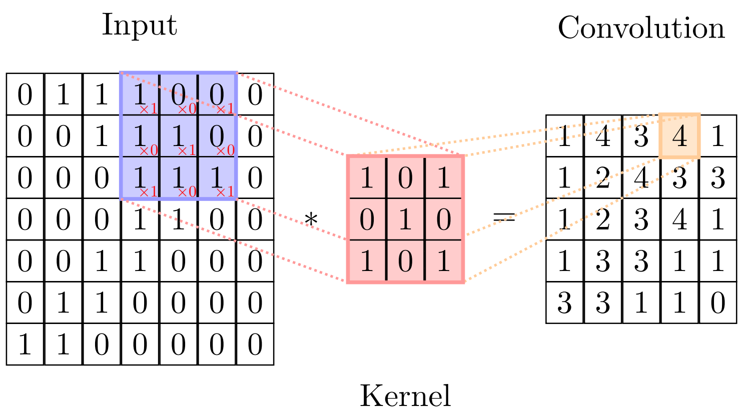


Figure 2.1.5: An example of a convolution operation applied to an input matrix.

While the subject of Deep Learning is far more comprehensive than this summary leads on, the introductory background will be sufficient to introduce and understand the models used in the rest of this work. As we now take strides towards addressing the subject of *Deep Reinforcement Learning*, our attention first shifts to cover the fundamentals of *Reinforcement Learning*.

2.1.2 Reinforcement Learning

The principle objective behind any Reinforcement Learning application is to teach an agent, capable of interacting with her environment, how to perform an outlined task optimally by maximizing some notion of a cumulative reward. The setup is captured by a Markov Decision Process (MDP), as specified by the attributes $(\mathcal{S}, \mathcal{A}, r, p, \nu)$, where:

- $\mathcal{S}$ is a finite set providing the *State Space* for the environment;
- $\mathcal{A}$ is also a finite set giving the *Action Space*, which define the valid interactions that the agent may take with the environment;
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward signal which scores the result of an action against some heuristic;
- $p : \mathcal{S} \times \mathcal{A} \to [0, 1]$ denotes a transition probability where

  $$p_a(s, s') = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$$

  measures the probability of transitioning from state s to s' after performing action a at time t;
- $\nu : \mathcal{S} \to [0, 1]$ provides the initial probability distribution for each state.

Together, these ingredients define an environment and specify how the permitted interactions can be made. The mechanism driving the agent's interactions with the environment is referred to as a policy function $\pi : \mathcal{S} \rightarrow \mathcal{A}$, and is parameterized by domain-specific data θ. The end-to-end process for Reinforcement Learning is visualized in Figure 2.1.6 below:

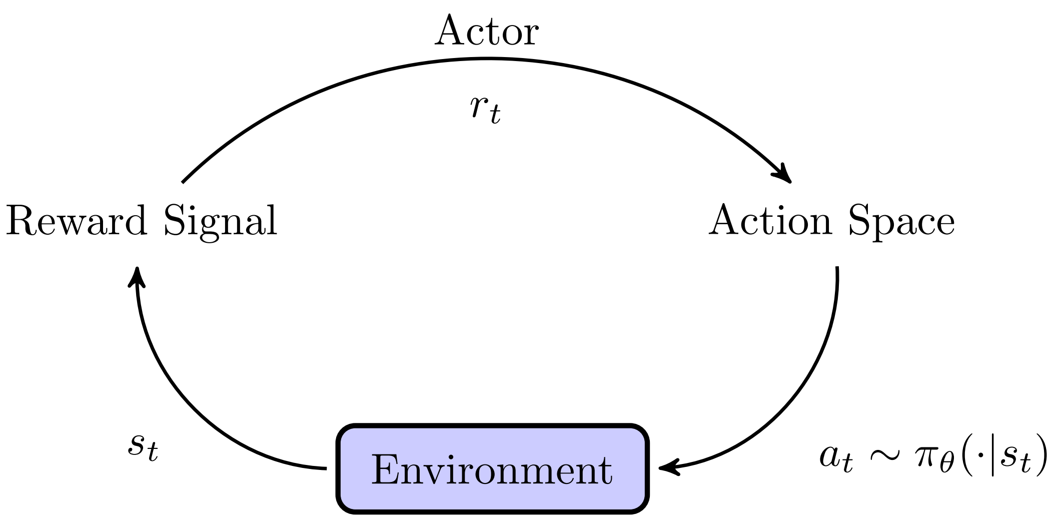


Figure 2.1.6: A simple model for a Reinforcement Learning process.

When the agent acts according to $a_t \sim \pi_\theta(\cdot|s_t)$ over the time horizon τ, the collection of actions form a trajectory $X = \{a_0, a_1, \ldots, a_\tau\}$. Over a given trajectory, the agent interacts with her environment and receives feedback in the form of a reward signal. The reward signal plays an important role in providing an incentive for the agent to improve her performance. To this effect, for a given trajectory X, the cumulative discounted return is

$$R(X) = \sum_{t=1}^{\tau} \gamma^t r_t$$

Where $\gamma \in [0, 1]$ is a discount factor applied to future rewards. The basic premise behind applying a discounting factor is that immediate returns should be prioritized over delayed returns. From a mathematical perspective, keeping $\gamma \in [0, 1]$ also helps ensure convergence with many Reinforcement Learning algorithms, especially in the cases when $\tau \rightarrow \infty$ as with continual-learning agents.

For a trajectory X consisting of finite-steps up to τ, the probability of the occurrence is given by the Markovian τ-step transition probability

$$\mathbb{P}(X|\pi) = \nu(s_0) \prod_{t=0}^{\tau-1} \mathbb{P}(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \tag{2.1.1}$$

Which then immediately facilitates the expectation of the cumulative discounted returns being

$$\mathbb{E}_{X\sim\pi}\Big(R(X)\Big) = \int_X \mathbb{P}(X|\pi)R(X) \tag{2.1.2}$$

The expected cumulative return may very well become the subject for maximization with respect to the policy π. This, in effect, works to optimize the decisioning of an agent following the policy, so as to accumulate the greatest sum of returns. Exactly how this expectation is maximized can be viewed from two perspectives. For the optimal policy π^*, the optimal value function V^* and the optimal action-value function Q^* are expressed

$$V^*(s) = \max_\pi \ \mathbb{E}_\pi\Big(R(X)\Big|s_0\Big) \tag{2.1.3}$$

$$Q^*(s,a) = \max_\pi \ \mathbb{E}_\pi\Big(R(X)\Big|s_0, a_0\Big) \tag{2.1.4}$$

Intuitively Equation 2.1.3 and Equation 2.1.4 differ in that Q^* incorporates the initial action taken by the agent. It follows that $V^*(s) = \max_a Q^*(s,a)$. In light of this property, both V^* and Q^* respect an important self-consistency principle that comes from classical Control Theory. The term "optimal control" arose in the 1950's when researchers began focusing on minimizing functions which describe dynamical systems [71]. One of the main contributions Control Theory came from Richard Bellman [70], who introduced the Dynamic Programming (DP) that outlines a recursive formulation under which Equation 2.1.3 and Equation 2.1.4 may be written

$$V^*(s) = \max_a \ \mathbb{E}_{s'\sim p}\Big(r(s,a) + \gamma V^*(s')\Big) \tag{2.1.5}$$

$$Q^*(s,a) = \ \mathbb{E}_{s'\sim p}\Big(r(s,a) + \gamma \max_{a'} Q^*(s',a')\Big) \tag{2.1.6}$$

Equation 2.1.6 states that under an optimal policy, the optimal action-value can be decomposed into an immediate reward contribution plus the benefit yet to be realized over the remaining trajectory. The DP Principle applies directly to discrete systems, whereas the continuous analog takes the form of a partial differential equation referred to as the Hamilton-Jacobi-Bellman (HJB) equation [46].

Reinforcement Learning calls for a balance between state-exploration and reward-exploitation. The principle objective remains to maximize the expected rewards; however, the agent needs to first accumulate a wealth of experience interacting with her environment in order to learn how to act optimally. Solving the exploration/exploitation tradeoff is at the heart of all Reinforcement Learning applications, and was solved for the classic "Multi-arm Bandit" example by John Gittins in 1979 [33].

2.1.2.1 Policy Optimization versus Q-Learning

When performing Reinforcement Learning, the learning objective may be supported by either optimizing policies or Q-functions [22]. Policy optimization tackles the learning task directly by optimizing the parameters θ which maximize the expected performance of π_θ. Policy optimization usually involves computing estimates for V^*, but other approaches involving surrogate functions may also be adopted [79] which can be lightweight and easily scaled.

On the other hand, Q-Learning is supported by estimating approximators for the optimal action-value function Q^*. Then, by applying the Bellman Principle (Equation 2.1.6), iterative updates motivate optimal decisioning from the agent that result from the actions which maximize the Q-function, $a_t = \arg\max_a Q^\pi(s, a)$. Q-Learning gives rise to one of the most famous and conceptually important Reinforcement Learning algorithms, and is presented below:

Algorithm 1 Q-Learning [92]

```
Initialize $Q(s,a)$ and $\pi$ randomly
repeat
  Initialize $s$
  repeat
    Choose $a$ from $s$ using $\pi$
    Take action $a$, collect $r$, observe $s'$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma \max_a Q(s',a) - Q(s,a)\Big)$
    $s \leftarrow s'$
  until $s$ is terminal
until no episodes remaining
```

Policy optimization is usually conducted *on-policy*, whereas Q-Learning, as given above, is an *off-policy* approach. The difference between an approach being on- and off-policy results from how the data, pertaining to either the action-values or the policy itself, is used to develop subsequent updates. Specifically, on-policy methods utilize the most recent version of the policy whereas off-policy methods utilize data collected over the entire trajectory. An *experience replay buffer* may be introduced to adapt an on-policy method into a hybrid approach, such as the case with [15]. Utilizing an experience replay buffer has become a common approach in various *Deep Reinforcement Learning* adaptations of Q-Learning, to help bring about stability and improvements when prioritizing learning experiences.

2.1.2.2 Generalized Advantage Estimation

We revisit policy optimization with the objective of reviewing a family of policy gradient estimators. As an item of notation, we will express the gradient operator with respect to parameter θ as ∇_θ. Then, the gradient of Equation 2.1.1 may be written

$$\nabla_\theta \mathbb{P}(X|\pi) = \mathbb{P}(X|\pi)\nabla_\theta \log \mathbb{P}(X|\pi)$$

And thus, the gradient of Equation 2.1.2 becomes

$$\begin{aligned}\nabla_\theta \mathbb{E}_{X\sim\pi}\Big(R(X)\Big) &= \int_X \nabla_\theta \mathbb{P}(X|\pi)R(X) \\ &= \int_X \mathbb{P}(X|\pi)\nabla_\theta \log \mathbb{P}(X|\pi)R(X) \\ &= \mathbb{E}_{X\sim\pi}\Big(\nabla_\theta \log \mathbb{P}(X|\pi)R(X)\Big) \\ &= \mathbb{E}_{X\sim\pi}\Big(\sum_{t=0}^{\tau-1} \nabla_\theta \log \pi_\theta(a_t|s_t)R(X)\Big)\end{aligned}$$

With the gradient in hand, the parameters θ are optimized by performing the following gradient *ascension*

$$\theta \leftarrow \theta + \alpha\nabla_\theta \mathbb{E}_{X\sim\pi}\Big(R(X)\Big)$$

Whereas the formulation shown above uses discounted rewards $R(X)$ as an incentive, other functions such as the *advantage function* A^π, offers related performance-incentivizing interpretations. The advantage function is defined as the difference between the action-value function Q^π and the value function V^π,

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

Which scores the benefit of performing a particular action given all of the possibilities that the agent could have chosen in the current state. Utilizing the advantage function has become common for defining policy estimators for many *Actor-Critic* algorithms, such as the one we will soon introduce. When it comes to estimating the advantage function, the *Generalized Advantage Estimator* (GAE) was introduced in [78], and is defined as

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{k=0}^{\infty}(\gamma\lambda)^k\Big(r_{t+k} + \gamma V^\pi(s_{t+k+1}) - V^\pi(s_{t+k})\Big)$$

For parameter $\lambda \in [0, 1]$. The estimation of the advantage function above then directly facilitates taking a sample mean to estimate the policy gradient $\nabla_\theta \mathbb{E}_{X\sim\pi}\Big(A^\pi(X)\Big)$.

2.1.2.3 Asynchronous Advantage Actor Critic

The work to introduce policy gradient estimators, and specifically the GAE, was done with the intention of making a connection to the Asynchronous Advantage Actor Critic (A3C) algorithm [90]. The A3C algorithm, like other Actor-Critic models, maintains both a policy estimate and a prediction for the value function. Where A3C differs from other approaches is in the support for multiprocessing; each sub-process inherits global parameters for the policy and value function, and independently gathers experience to inform updates on the shared parameters. Each sub-process then asynchronously relays parameter updates back to the global model, and the cycle begins anew. A3C was introduced on an n-step ahead basis, providing each Actor-Critic sub-process with an n-step window to collect rewards and values, as aggregated to provide an estimate for the advantage function. The A3C algorithm is shown below in Algorithm 2. A diagram of the overall A3C routine is also given in Figure 2.1.7.

A3C models tend to observe reduced training times over non-parallel Actor-Critics, with a time reduction that is approximately linear in the number of sub-processes. The authors of [90] also remark on, and observe, the improvements to learning-stability which is brought forth by having multiple agents independently learn policy and value updates. Also from a computational resource perspective, A3C runs efficiently on a computing device with multiple dedicated cores, and for this reason, is well-suited for general use over algorithms which are best performed under a GPU. For this work, the A3C model has been adapted to run for both CPU-only machines *and* instances with multiple GPUs.

Algorithm 2 Asynchronous Advantage Actor Critic [90]

```
Inherit shared global parameters $\theta$, $\theta_v$ and common global counter $T = 0$
Initialize process-specific parameters $\theta'$, $\theta'_v$
Initialize process step counter $t \leftarrow 1$
repeat
    Reset gradients $d\theta \leftarrow 0$, $d\theta_v \leftarrow 0$
    Synchronize process-specific parameters $\theta' \leftarrow \theta$, $\theta'_v \leftarrow \theta_v$
    $t_{\text{start}} \leftarrow t$
    repeat
        Choose $a_t$ from $s_t$ using $\pi$
        Take action $a_t$, collect $r$, observe $s_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until $s_t$ is terminal or $t - t_{\text{start}}$ reaches the maximum step count
    $R = \begin{cases} 0 & s_t \text{ terminal} \\ V(s_t, \theta'_v) & \text{otherwise} \end{cases}$
    $i \leftarrow t - 1$
    repeat
        $R \leftarrow r_i + \gamma R$
        $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')\Big(R - V(s_i, \theta'_v)\Big)$
        $d\theta_v \leftarrow d\theta_v + \frac{\partial}{\partial \theta'_v}\Big(R - V(s_i, \theta'_v)\Big)^2$
        $i \leftarrow i - 1$
    until $i < t_{\text{start}}$
    Asynchronous updates of shared global parameters
until $T >$ maximum time for the episode
```

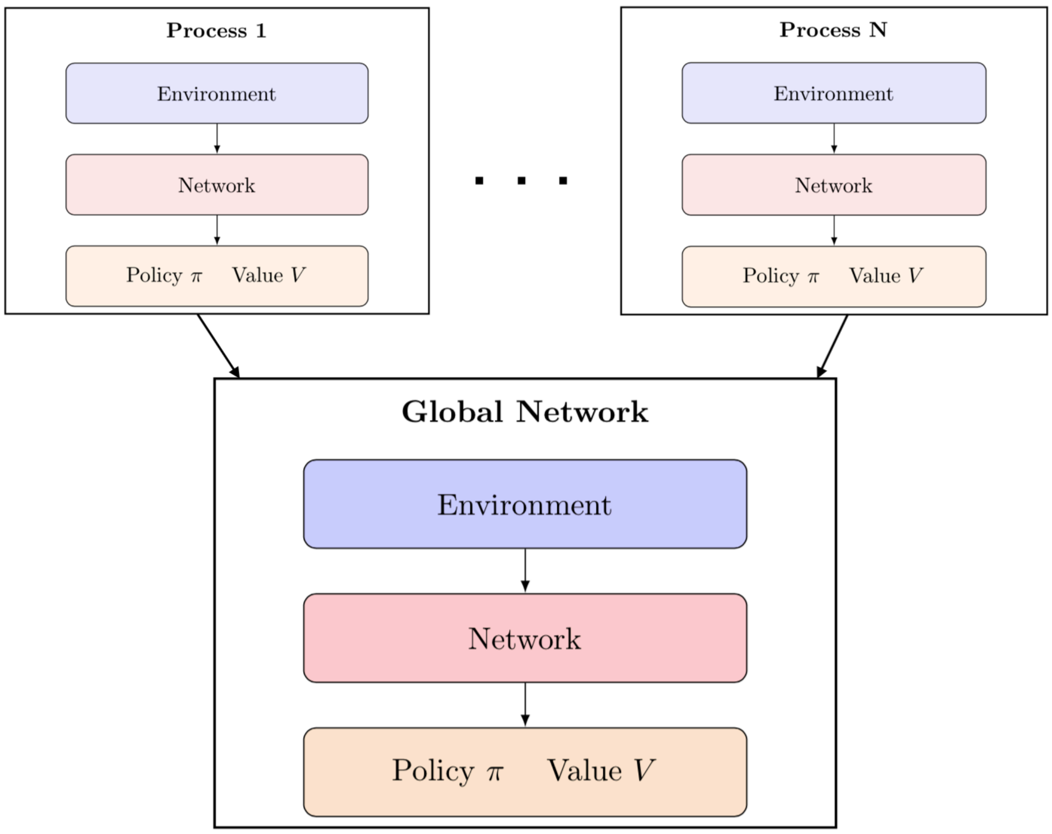


Figure 2.1.7: Diagram of an A3C network.

Having covered A3C, the required background for Reinforcement Learning has been introduced. Our final discussion topics now turn to connecting the subjects of Deep Learning with Reinforcement Learning and then motivating examples of Transfer Learning.

2.1.3 Deep Reinforcement Learning

The subject area Deep Reinforcement Learning comes about when Deep Learning solutions are utilized to facilitate a Reinforcement Learning objective. Among the most influential contributions to Deep Reinforcement Learning came from the introduction of *Deep Q-Networks* (DQN) [58], which was the first approach that utilized a Neural Network to approximate the action-values used in Q-Learning (Algorithm 1). At the time of introduction, DQN achieved state-of-the-art performance on six Atari 2600 arcade games with “no adjustment of the architecture or hyperparameters” [58]. The performance of DQN, in combination with the lack of required network tuning, firmly demonstrated that Deep Learning can successfully facilitate Reinforcement Learning. DQN however was not without flaws, particularly in a tendency to overestimate action-values, thus motivating the following development which introduced *Double-Deep Q-Networks* (DDQN) [88]. DDQN uses an on-policy Neural Network to first select an action from which a second, off-policy Neural Network, scores the value used for parameter estimates. The addition of the second Neural Network reduces the bias and in turn, brings about greater stability and performance improvements over DQN.

Dueling Networks were introduced in [91] and split the typical DQN network into two components or "streams": the first stream estimates the state-value and the second estimates advantages for each action. Dueling Networks were considered better-suited adaptations for Reinforcement Learning tasks due to the architecture's support for learning values separately from learning actions. Dueling Networks can be supported with complementary practices in the field of Deep Reinforcement Learning, such as utilizing a *Priority Experience Replay* (PER) [76]. Experience buffers were briefly alluded to in section 2.1.2.1 as a way for on-policy Reinforcement Learning algorithms to leverage training experience in an off-policy way. However, not all the experience collected by an agent is useful and should be prioritized as such. This is the insight offer by PER, which presents a prioritization of the agent's experience based on the expected contribution to future learning.

There remains a wealth of literature that introduces new and proficient Deep Reinforcement Learning approaches, many of which build upon the successes of each other. To this tune, *Rainbow Learning* [40] was engineered with goal of hybridizing the methodologies from DDQ, Dueling-Networks, PER and Distributional Q-Learning [9], boasting record performances as justification.

Whereas all methods discussed so far attempt Deep Reinforcement Learning using a Q-Learning based approach, there are effective alternatives, such as *Proximal-Policy Optimization* (PPO) [79], that are adopted as standard algorithms due to their simplicity and ease in setup.

In this work, we select A3C as a base algorithm, driven by a Convolutional Neural Network with an LSTM to produce value and policy estimates. Our decision is supported on the account that we do not seek nor do we focus on state-of-the-art performance. Rather, our goal with the work in this chapter is to study the benefits of a Teacher/Student learning framework. As a result, we focus on learning algorithms which are stable and well-configured to operate on limited computational resources. Furthermore, the A3C approach conforms with the current implementation from the Bank, and we intend to present readily applicable results.

The last topic to address for background is an overview of *Transfer Learning*.

2.1.4 Transfer Learning

Transfer Learning is the task of training an agent to perform in a target environment by leveraging the experiences accumulated in a different environment. Similar to Deep Learning, Transfer Learning applies to a larger class of Machine Learning objectives, however within the context of this work, Transfer Learning will be viewed directly within the scope of Reinforcement Learning. Among the many sub-classifications, Transfer Learning can be approached from three perspectives: *Zero-shot Learning*, *Continual Learning* and *Curriculum Learning* [28].

In Zero-shot Learning, an agent attempts to learn generalizations between the policies learned from different environments. The general approach is to have an agent learn subtasks along with the corresponding dependencies to those subtasks. As shown in [83], a *Neural Subtask Graph* can be constructed to encode dependencies between subtasks and their propagated effects on the reward signal of a more complex task. The primary takeaway from [83] is that Transfer Learning can be approached from a structural perspective, which is a theme that will inform the models we develop.

One of the central applications to Zero-shot Learning comes from the crucial and necessary adaptation between simulated and real-world environments. Deep Reinforcement Learning often can only be conducted under simplified or simulated conditions; however, this may not be conducive to real-world systems which are inherently more complicated. Navigating the difficulties between simulations and real-world systems using Zero-shot Learning is studied in [32] and [94], and becomes relevant to section 4.2 where a simulated environment is used to train a Deep Reinforcement Learning model for Order Execution. To this extent, we subsequently revisit the notion of Zero-shot Learning when we discuss the design of the training environments conducive for Order Execution.

The Continual Learning of an agent refers to the agent's ability to learn new tasks whilst preserving judgement and decisioning on previously collected experience. Continual learning attempts to design "life-long learners" which embody the concept that enriched learning is cumulative. The task of continual learning is complicated by what's known as *catastrophic forgetting* [65]. A model catastrophically forgets if the updated weights bring about a departure from the agent achieving optimal decisioning on their previously learned policy. Advice from [65] indicates that catastrophic forgetting may be overcome if an agent is exposed to new learning material in a scheme that does not interfere with existing knowledge. Such sentiments echo in [52]; the findings from which indicate that catastrophic forgetting can be mitigated by tuning the speed of learning for the weights which are deemed important to the central task. The adjustment of the weights is based on an *Elastic Weight Consolidation* (EWC) and involves modifying the policy loss function so as to penalize large departures in the updated weights from that of the previous model. In a sense acting like a regularizer, EWC ensures that the weights remain similar to the previously learned model whilst still allowing for updates to facilitate the new task effectively.

Curriculum Learning is realized when an agent is trained with input from an advising source. The goal is to reduce the time it takes the agent to acquire proficiency on a target task. There are several approaches for Curriculum Learning, including *Teaching on a Budget* for Deep Reinforcement Learning from [43], where a Teacher is optimized to provided advice on a limited basis, for a Student attempting to learn a task. Another interpretation of *transfer advice* comes from [87], where a knowledge mapping is passed from a Teacher to a Student in the form of a support vector regression.

In the work to follow in section 2.2, we cast Continual and Curriculum Learning models, under which a Teacher model receives the objective *to stay alive for as long as possible* within the environment, and the Student model receives the objective *to earn the highest reward possible* within the environment. We primarily approach the Transfer Learning objective structurally, designing two Neural Networks that incorporate Teacher advice and state observations under different constructs. We also draw on standard methods in Transfer Learning and initialize our Student models with the pretrained network weights from the Teacher, as done in [50] and [95].

2.2 Methodology

We begin this section with a brief technical note regarding our computing resources, followed by an overview of the three Atari 2600 arcade games used for training. For comparison purposes, a baseline model is introduced that will outline common elements for the subsequent Teacher/Student models, including the network architecture and the tuned hyperparameters.

The core objective behind the Teacher/Student framework is to achieve accelerated learning with the Student agent by processing advice from the Teacher agent. The Teacher and Student models each receive different objectives, so we first introduce the training scheme for the Teacher. The model for the Teacher has the same network structure as the baseline model, but the Teacher is incentivized to *stay alive for as long as possible.* We provide a modified reward signal that reinforces the Teacher to learn this survival objective.

The Student models are trained to the specifications of the original Atari 2600 arcade games. Subsequently, two Student learning models are presented based on different network structures that facilitate *Advice-Driven* (AD) decisioning and *Advice & State-Driven* (ASD) decisioning with respect to the Teacher. In the case of the ASD model, the network structure allows for the weights to be preinitialized based on the trained parameters from the Teacher.

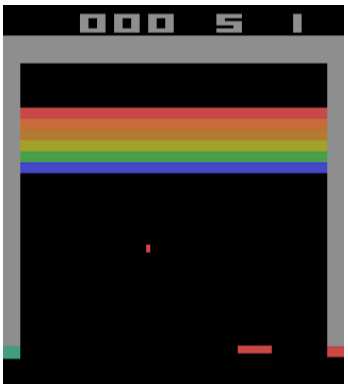
We conclude by providing a remark on what performance metrics will be used to study the effectiveness of the Teacher/Student framework.

2.2.1 Computing System Configurations

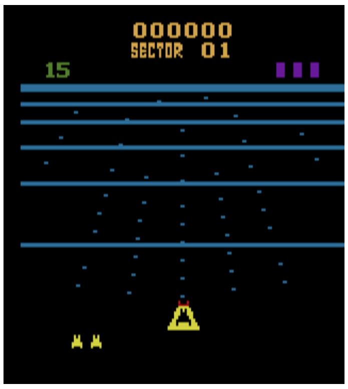
All models in this chapter are trained using a dedicated Virtual Machine instance equipped with 16 vCPUs, 104GB of memory, a 100GB Standard Persistent Disk, and 4 NVIDIA Tesla K80 GPUs. Correspondingly the A3C model, with the corresponding utilities, are forked from a public Github Repository[1] [36]. We appreciate that many of the configurations we adopt follow from that provided in this repository. We express our gratitude to the author for publishing the code as open-source material.

2.2.2 The Atari 2600 Environment

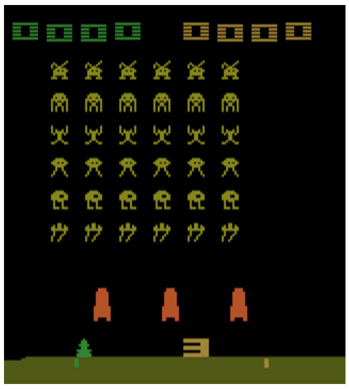
Emulators for classic Atari 2600 arcade games are commonly used as benchmark training environments for which Deep Reinforcement Learning models can be studied, tuned and enhanced [56]. The emulators for the Atari 2600 environments are provided by OpenAI's *Gym* [16], and are well suited for the upcoming Teacher/Student learning models. Shown below in Figure 2.2.1 are screenshots of the initialized, and preprocessed, states for the Atari 2600 arcade games covered.



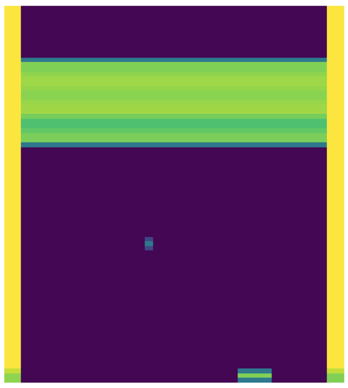
(a) Breakout.



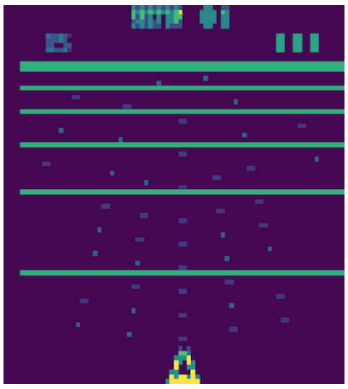
(b) Beamrider.



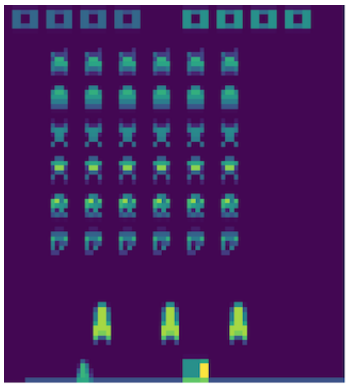
(c) Space Invaders.



(d) Processed Breakout.



(e) Processed Beamrider.



(f) Processed Space Invaders.

Figure 2.2.1: Screenshots of three Atari 2600 arcade games.

[1]The repository originates from `https://github.com/dgriff777/rl_a3c_pytorch`

The action space for each Atari 2600 environment is discrete and corresponds to the degree of freedom for the game. To this effect, there are four actions available in Breakout, nine in Beamrider and six in Space Invaders.

For each Atari 2600 environment, an observation of the state space is an RGB image of dimension $(210, 160, 3)$. To reduce computational overhead, the observations are preprocessed by condensing the image to the average pixel-value, down-sampling the pixel-density by a factor of 255, and then finally casting the dimension to be $(1, 1, 80)$. In addition to reducing the complexity of the input, image-preprocessing carries the benefit of focusing the observation-space to the features which are most important to facilitate task-learning. Preprocessing has also become a commonly adopted practice with the feature engineering for convolutional networks [82], which the one we will now introduce derives from.

2.2.3 Baseline Model

We introduce a baseline model for the purpose of establishing benchmark performance results which can be compared to the performance of the Teacher/Student learners. The baseline model will also come to define the network structure that the Teacher model inherits. Presented below in Figure 2.2.2 is the network.

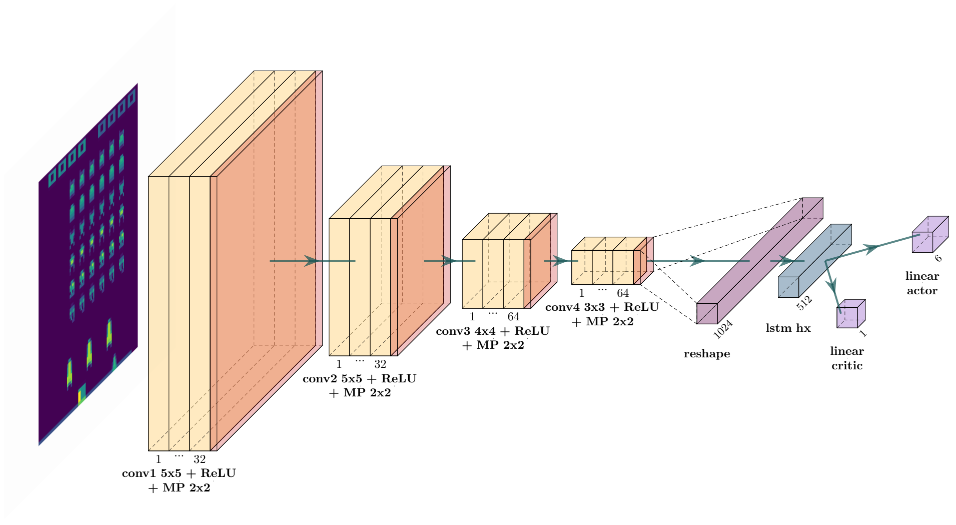

1 ⋯ 32
conv1 5x5 + ReLU + MP 2x2
1 ⋯ 32
conv2 5x5 + ReLU + MP 2x2
1 ⋯ 64
conv3 4x4 + ReLU + MP 2x2
1 ⋯ 64
conv4 3x3 + ReLU + MP 2x2
1024
reshape
512
lstm hx
1
linear critic
6
linear actor

Figure 2.2.2: The baseline model used for Space Invaders, from [36].

Remark 2.2.1 *Each component in Figure 2.2.2 is interpreted as the resulting tensor after passing through the labelled portion of the network. For example, the first thirty-two dimensional structure is the output result after a two dimensional convolutional layer – with a (5×5) kernel, receiving ReLU activation and a (2×2) max-pooling – is applied to the input image. All convolutional layers are given a padding of one, with the exception of the first convolutional layer, which receives a padding of two.*

Table 2.2.1: Common hyperparameter configurations.

A3C Parameters	
Discount Factor γ	0.99
GAE Factor λ	1.00
Number of Steps	20
Number of Processes	16

Environment Parameters	
Maximum Episode Length	10,000
Frameskip Rate	4

Optimization Parameters	
Learning Rate α	10^{-4}
Optimizer	Adam
Shared Optimizer	True
Amsgrad	True

With the processed-state observation as an input, the network for the baseline model is characterized by four sequential convolutional layers followed by an LSTM cell that outputs approximations for the agent's policy and value function respectively. Note that Figure 2.2.2 presents the baseline model as it applies to Space Invaders – the only modification made for Breakout and Beamrider is with the input image and the dimension of the *Actor*'s output, matching that of the cardinality for the respective action space.

The A3C algorithm (Algorithm 2) is adapted to train the network, where the advantage function is estimated using GAE. The remaining hyperparameters are shown in Table 2.2.1, and are kept constant throughout the training of all models. The tuning of these hyperparameters is done minimally, forming the basis for one of the suggestions for future work presented in section 2.4. However, a discount factor γ being close to one is justified on the account of keeping the models "far-sighted".

Having understood the baseline model, the training scheme for the Teacher model is now introduced.

2.2.4 Teacher Training Scheme

The Teacher model inherits the same network structure as the baseline model. Where the training scheme differs is in the objective assigned to the Teacher. Formally, the Teacher is trained to learn a policy which ensures that she continually interacts with her environment *and survives* for as long as possible.

We modify the reward signal to encourage the Teacher to learn a survival strategy. Presently, the following reward signal is crafted and an intuitive justification is provided thereafter. The reward signal that reinforces the Teacher is taken as

$$r(t) = \frac{t\eta}{t_{\max}}\Big(1 + \mathbb{1}_{\{\texttt{done}\wedge\texttt{cleared}\}}\Big) - \kappa\mathbb{1}_{\{\texttt{done}\wedge\neg\texttt{cleared}\}} \tag{2.2.1}$$

Where t is the episode length, $t_{\max}$ is the maximum episode length taken as 10,000 as per Table 2.2.1, and parameters η, κ are interpreted respectively as the survival bonus and the hit penalty. The two indicator functions are drawn from the binary events `done` and `cleared`. The event `done` is true whenever the environment is reset, which may be triggered when the agent fails her task *or* successfully clears a stage; to distinguish between the possible cases, `cleared` being true indicates that the agent was successful in reaching a terminal state. The distinction is really only important for Space Invaders and Beamrider, since both games naturally continue in a reset state if the agent clears all the enemies on the screen, and is otherwise a subtle technicality. For the implementation to follow, we took $\eta = 1/2$ and $\kappa = 1$; extensions of this reward signal may choose different, and perhaps non-constant, values for these parameters.

The reward signal is similar to, and indeed inspired by, the unit-reward signal from the classic control game "CartPole"[2]. In Cartpole, an agent is tasked with balancing an upright pole on a moving cart, and is scored only on the duration over which the agent can keep the pole upright whilst the cart moves. CartPole is a very simple game in terms of the action and state spaces, and for this reason a unit-reward signal is an appropriate measure to motivate the agent to survive for as long as possible. However, for comparatively more complicated environments, such as the three Atari 2600 arcade games attempted in this work, a stronger correlation between reinforcement and action is required. Specifically, Equation 2.2.1 is crafted to increase when the Teacher learns to survive for longer periods of time. Empirical tests of the unit-reward signal, specifically in reference to Figure 2.4.1, demonstrate a significantly reduced rate of learning. We interpret this as indication that the unit-reward signal offers too weak of a correlation between action and reinforcement; from "over-encouraging" the Teacher, she becomes unable to draw inference and meaningful interpretation from the actions taken.

We demonstrate that the Teacher can learn a survival objective by making the reward signal an increasing function of the episode length. The primary performance metric to evaluate the success of the Teacher model is the average episode length over a training session. The reasonable expectation is that a proficient Teacher should realize an average episode length that approaches the maximum episode length of $10,000$. This indeed is the case for the Teacher models trained on Breakout and Beamrider; however, the Teacher trying to survive in Space Invaders did not achieve such a feat. We recognize that this suggests deficiency in the crafted reward signal, and in section 2.4 we call for future development.

[2]https://gym.openai.com/envs/CartPole-v0/

2.2.5 Student Training Scheme

By introducing the Student models, the following concepts borrow from topics in Transfer Learning as discussed in subsection 2.1.4. The first model is a curriculum-learning agent that is fed advice directly from the Teacher agent, and receives no other form of observation on the state. Such a learning approach is henceforth referred to as *Advice-Driven* (AD) decisioning. The second model facilitates an advice stream from the Teacher *in addition to* a state observation; the result is the *Advice & State-Driven* (ASD) decisioning process.

The Student models are trained on the original Atari 2600 arcade games, and otherwise receive no modifications thereof. The training is then consistent with that of the baseline model, allowing for direct comparisons to drawn. Naturally, this also involves keeping the hyperparameters configured identically as per Table 2.2.1.

2.2.5.1 Advice-Driven Decisioning

The model for the AD learner is given in Figure 2.2.3. The input corresponds to advice from the Teacher, and is sourced by "detaching" the output state of the Teacher's LSTM cell. In this sense, the Student model receives advice based on the collective experiences of the Teacher. The Student preserves an ability to learn a policy on the environment she is exposed to by generalizing the advice received from the Teacher.

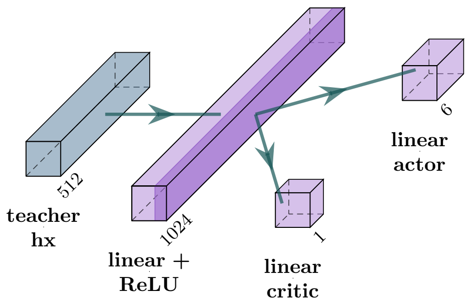


Figure 2.2.3: The Advice-Driven model structure.

Intuitively, an AD model is expected to perform well when the Teacher's task is similar to the Student's objective. Since the AD model does not directly process a state observation, decisions rely on a second-hand connection to the environment and may suffer from poor generalization if the Teacher is suboptimally trained.

To introduce a first-hand connection to the environment is to add an observation stream. Accordingly, we introduce the next learning model which does just that.

2.2.5.2 Advice & State-Driven Decisioning

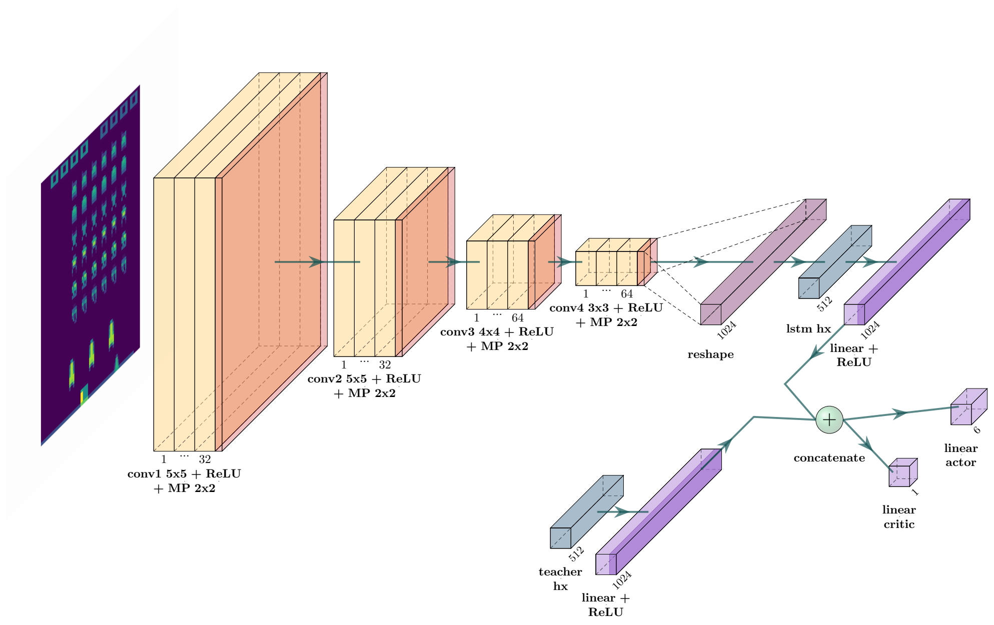


Figure 2.2.4: Advice & State-Driven model structure.

Figure 2.2.4 presents the network for the ASD learner. The model consists of an advice stream, identical to that of the AD learner, and an observation stream, similar to that of the baseline model with an added dense layer. The structure of the ASD network allows for the parameters in the observation stream to be initialized with corresponding weights from the Teacher's network. We refer to this practice as prescribing preinitialized weights (PIW). Under such scheme, the ASD learner would begin making informed decisions under the Teacher's directive and gradually shift her actions towards an optimal policy under the differing task.

We hypothesize that the ASD learner would better consume advice from the Teacher given the inclusion of the observation stream, ultimately contributing to an accelerated learning process.

2.2.5.3 Performance Measurement

We set the stage for section 2.3 by providing an overview of the performance metrics which will be used to interpret the output of the models. For the Teacher models, the main performance metric is the average episode length, which would indicate the extent to which the Teacher learned her survival objective.

The Student models are trained under their respective original Atari 2600 arcade environment, so naturally, we adopt the average episode reward as an evaluative performance metric. To verify expectations that a Teacher/Student framework accelerates training, we will pay attention to the episode count in regards to the Student's learning curves. It will also be insightful to study the average episode length for the Student learning models, to indicate how influential the Teacher's advice was for the Student.

2.3 Main Results

We introduce the results by first presenting the performance of the baseline models. Then the performance of the Teacher models are reviewed, subsequently followed by an analysis of the Student models.

2.3.1 Baseline Performance

We present the results for the baseline models here, which will be referenced in the commentary for the performance review of the Student models.

Breakout

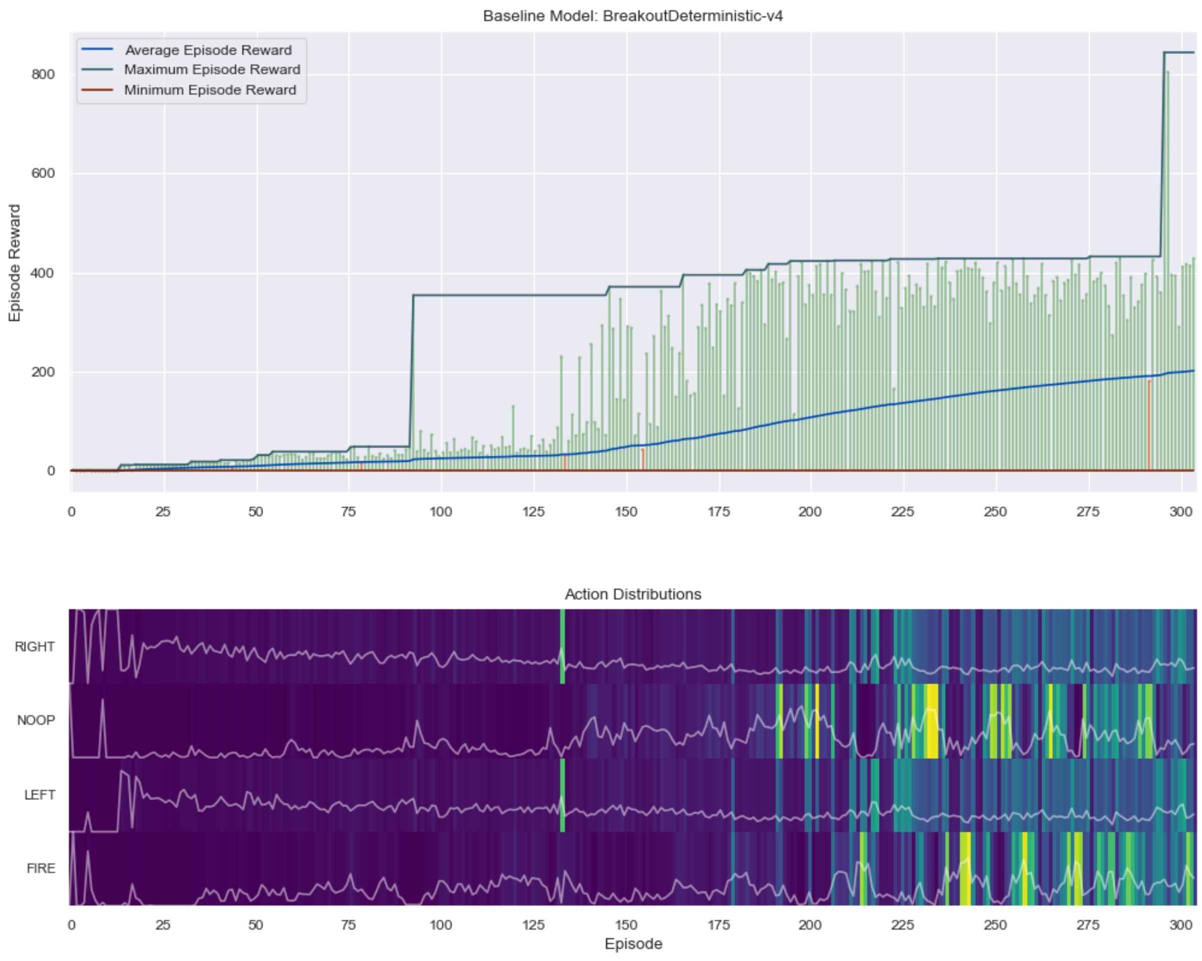


Figure 2.3.1: The performance of the baseline model for Breakout.

Figure 2.3.1 gives the learning curve for the baseline model on Breakout. In total, the training took three hours and 304 episodes were covered. Given the realized high-score towards the end of the session, it appears that the baseline model would continue to benefit from further training. In this light, we appeal to the following remark to make concrete our comparison approaches.

Remark 2.3.1 *Due to efforts to preserve computational resources on the virtual machine, we adhered to a strict training schedule. We do not anticipate this will cause any issues when we later compare the performance of the baseline models to the Student models, since ultimately we are attempting to demonstrate accelerated learning in the Student models. To account for the training horizon of the baseline models, we train the Student models under a similar session duration.*

Figure 2.3.2 gives the learning curve for the baseline model on Beamrider. The training also took three hours, spanning 278 episodes. Here, we observe a consistent and steady learning progression, with a slight levelling-off towards the end of the training session.

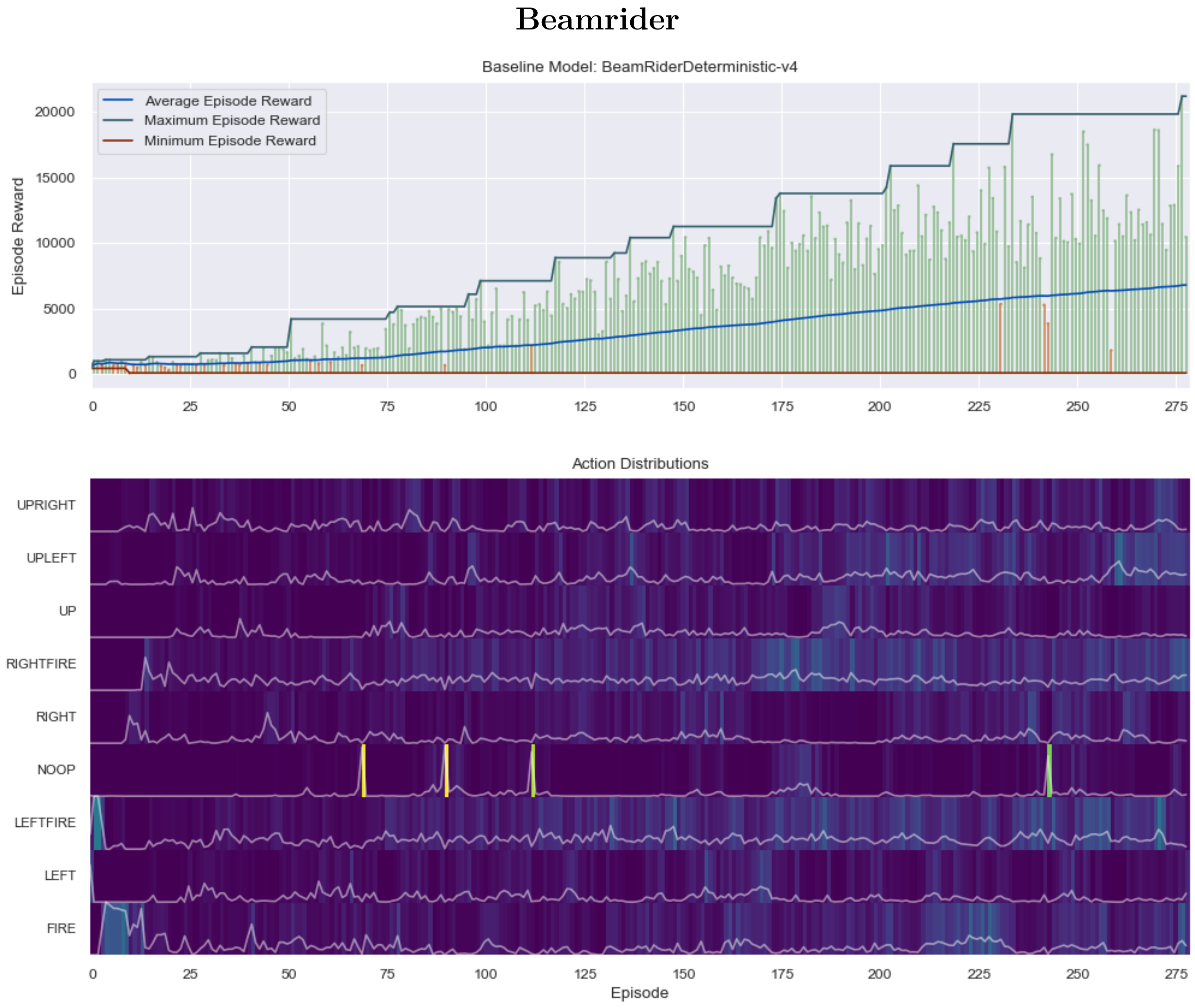


Figure 2.3.2: The performance of the baseline model for Beamrider.

Finally, Figure 2.3.3 shows how the baseline model for Space Invaders fared. We afforded a long training session lasting sixteen hours, elapsing 2256 episodes.

Space Invaders

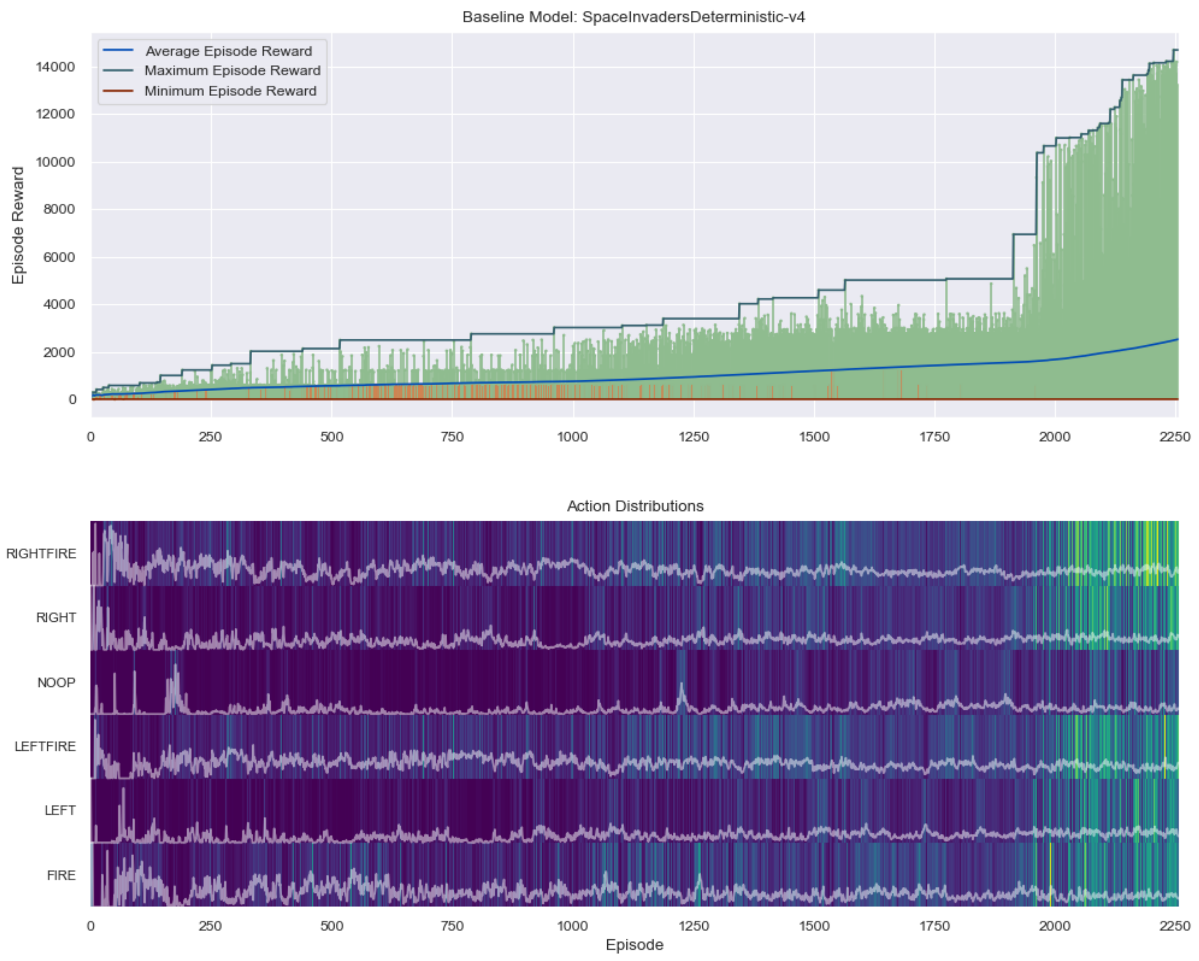


Figure 2.3.3: The performance of the baseline model for Space Invaders.

The long training session was apparently justified in the context of Space Invaders, since the baseline model drastically improved episode scores right at the end of the session. Which is to confirm preconceived notions that Space Invaders will be the most challenging learning environment. Accordingly, as we present results for the Teacher and Student models, we will afford more computational resources for Space Invaders, to reflect the findings from Figure 2.3.3.

Progressing forward to review the performances of the Teacher models.

2.3.2 Teacher Performance

Figure 2.3.4 gives the learning curve for the Teacher trained for survival on Breakout. The training spent 650 episodes over a three hour period, at which point, the Teacher was consistently surviving to the maximum episode length. This is not surprising, as Breakout is the simplest environment of the three Atari 2600 arcade games. Additionally, the survival objective is nearly identical to the task in the original game, suggesting that the Teacher model should be as proficient as the baseline model. We also take this as a sign that the Teacher model should provide valuable advice to the Student.

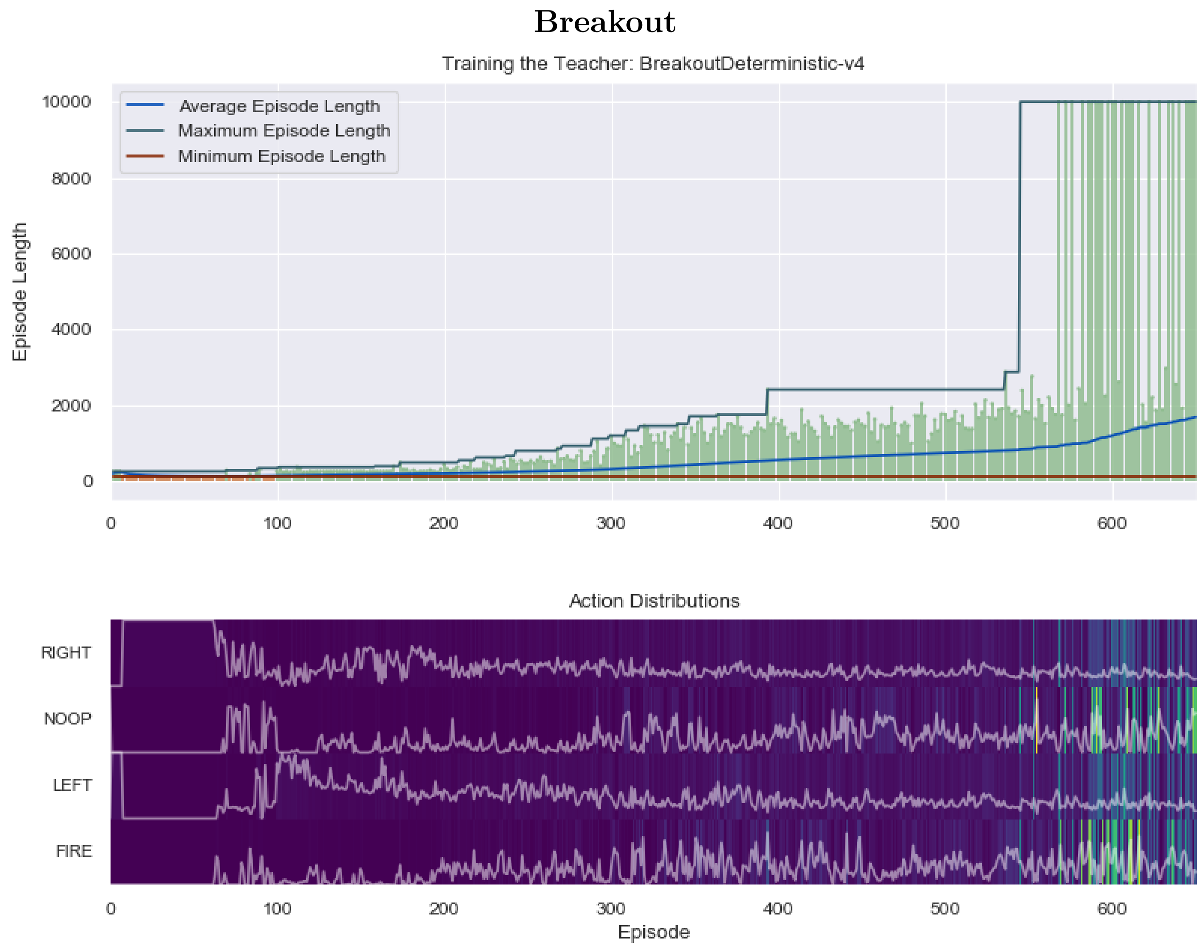


Figure 2.3.4: The performance of the Teacher model for Breakout. Data is plotted at every second training episode.

Figure 2.3.5 compares the action distributions of the Teacher and the baseline model. As expected, the actions of the Teacher very closely follow that of the baseline model and there is very little to distinguish the policies by. As we consider the more complicated environments of Beamrider and Space Invaders, we will begin to notice a clear depature in the policies of the baseline model and the Teacher.

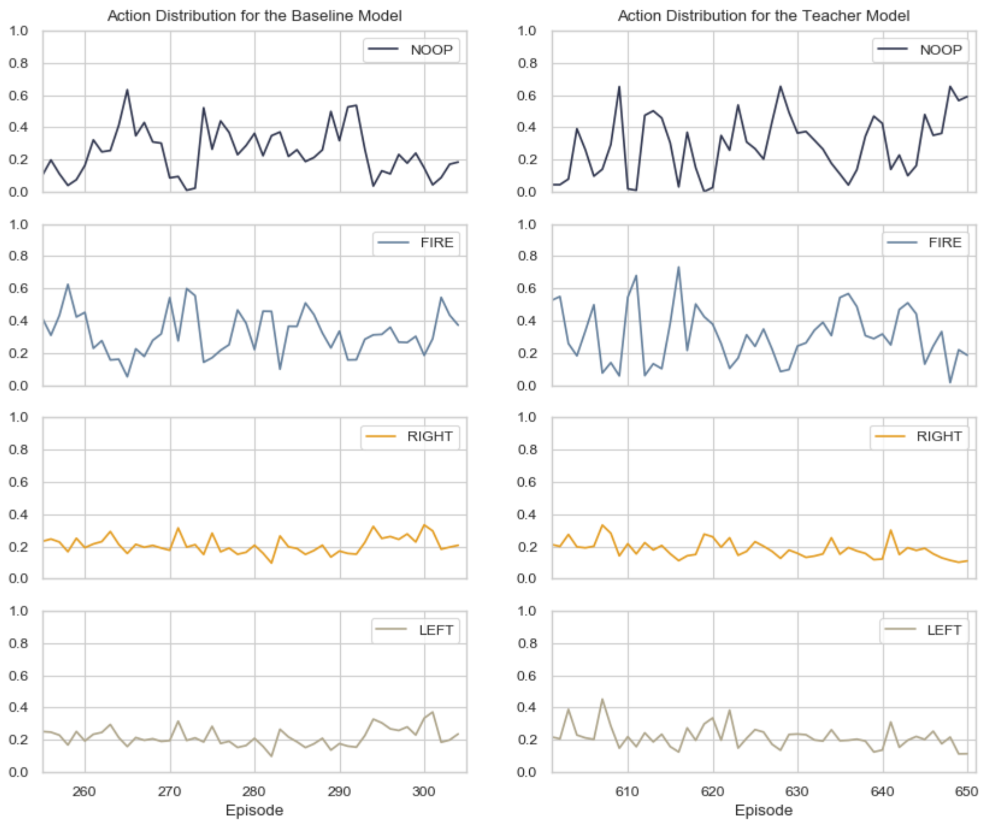


Figure 2.3.5: A comparative view of the action distributions between the baseline and Teacher models for Breakout. Data is displayed over the last fifty training episodes for each model.

Figure 2.3.6 presents the Teacher's learning curve for Beamrider. Training was performed for nine hours, which progressed through 940 episodes.

The tradeoff between exploration and exploitation is on clear display in Figure 2.3.6. The Teacher very early in her training obtained maximum performance by adopting a primarily defensive strategy. After initial success, there was a very clear drop in performance when the Teacher further explored her action space and began learning an offensive strategy. Of the three Atari 2600 games covered, Beamrider has the largest action space consisting of nine discrete actions. On account of experiencing the entirety of the action space, there are several periods when the performance of the Teacher drastically falls-off. The results indicate that the survival objective for Beamrider can be attained through different strategies, although the policy that the Teacher eventually settles on is one that is a mixture between an offensive and defensive strategy. Such a strategy is what we might expect from the baseline model, so we consult Figure 2.3.7 for further investigation.

Beamrider

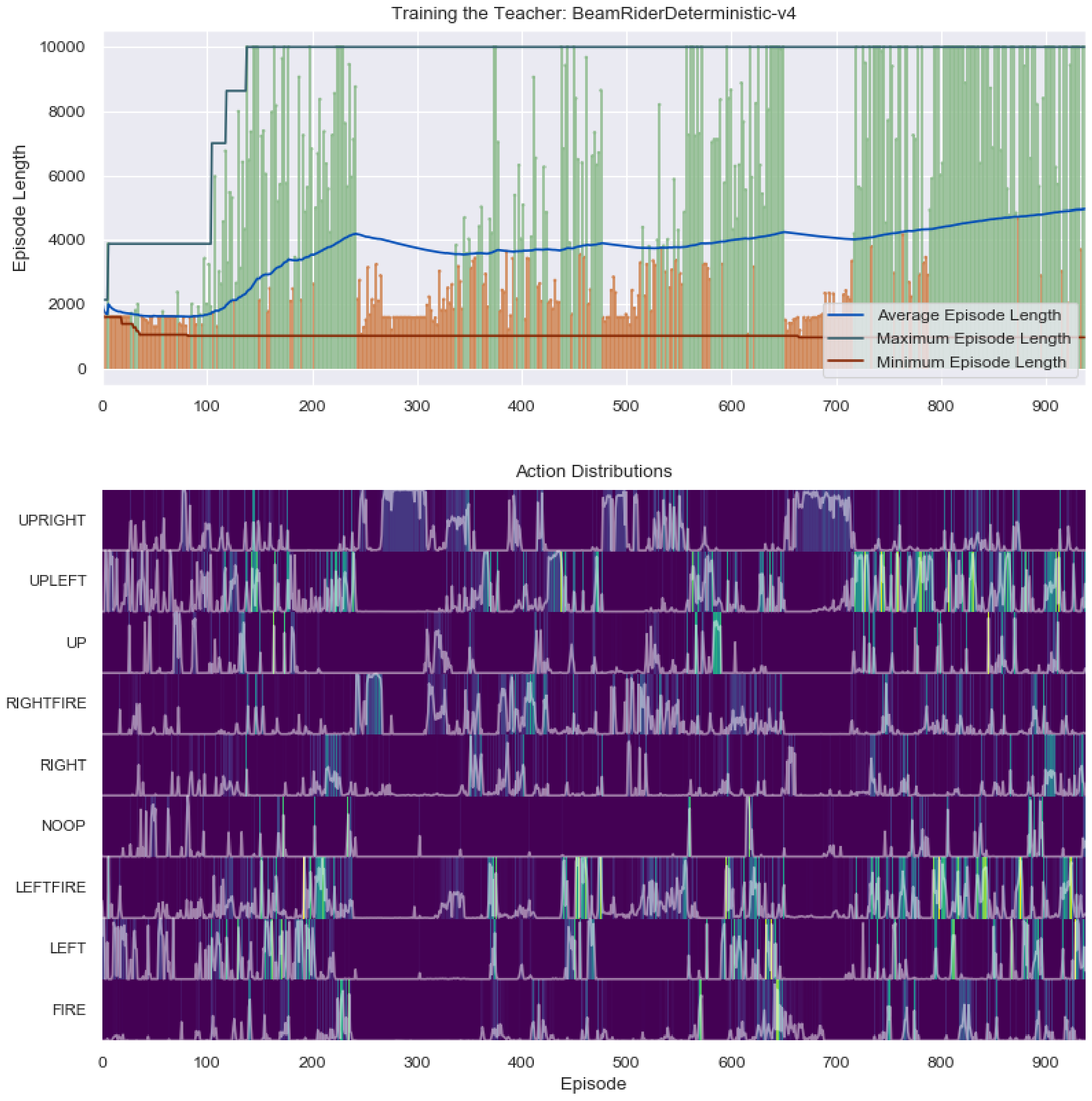


Figure 2.3.6: The performance of the Teacher model for Beamrider. Data is plotted at every second training episode.

In Figure 2.3.7, the policy decisions of the Teacher are noticeably less consistent. Whereas the baseline model captures a fairly steady split between an offensive and defensive strategy, the Teacher model tends more towards a single extreme.

Both Teacher models trained on Breakout and Beamrider were able to clearly demonstrate a policy that survives in their respective environment. The same result was not secured for Space Invaders, where at no point during the training session did the Teacher even come within fifty percent of the maximum target.

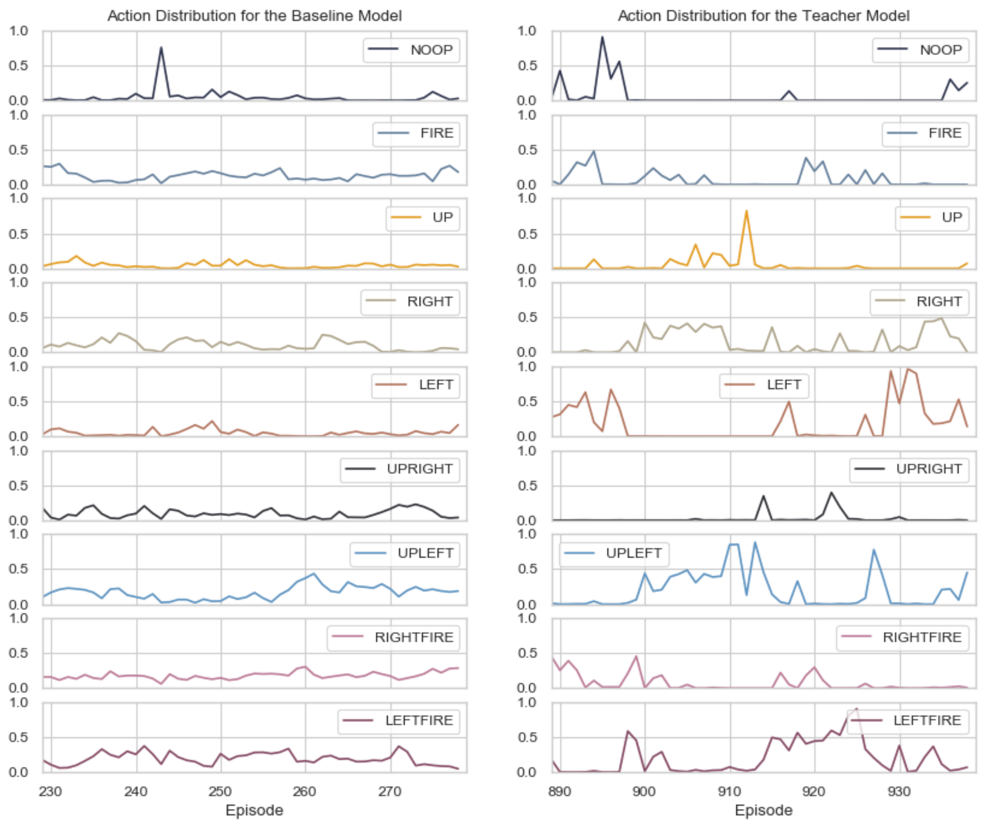


Figure 2.3.7: A comparative view of the action distributions between the baseline and Teacher models for Beamrider. Data is displayed over the last fifty training episodes for each model.

Finally, Figure 2.3.8 provides the learning curve for the Teacher trained on Space Invaders. The training took significantly longer; nineteen and-a-half hours elapsed in total, to cover 4100 episodes.

The action space for Space Invaders is less comprehensive than that for Beamrider, so the reduced performance cannot directly be attributed to the dimensionality of the action space. What's more, there is nothing that stands out from the action distributions that would suggest a failure of the Teacher in learning her objective. Rather, we suggest that the shortcomings of the Teacher trained on Space Invaders can be attributed to the nature of the game.

One possible explanation comes from the fact that Space Invaders, like Beamrider, is played over a series of *stages*. The difference is in the initialization of each stage: in Space Invaders, the enemies are reset to their starting positions whereas in Beamrider, the environment is cleared altogether. Such an initialization for the environment may pose a challenge for the Teacher drawing inference from observations on Space Invaders, resulting from a weak correlation between the reward signal and the presentation of the state. While ultimately we do not expect this to *prevent* learning entirely, it remains plausible that a more complicated state environment for Space Invaders slows the learning of the Teacher.

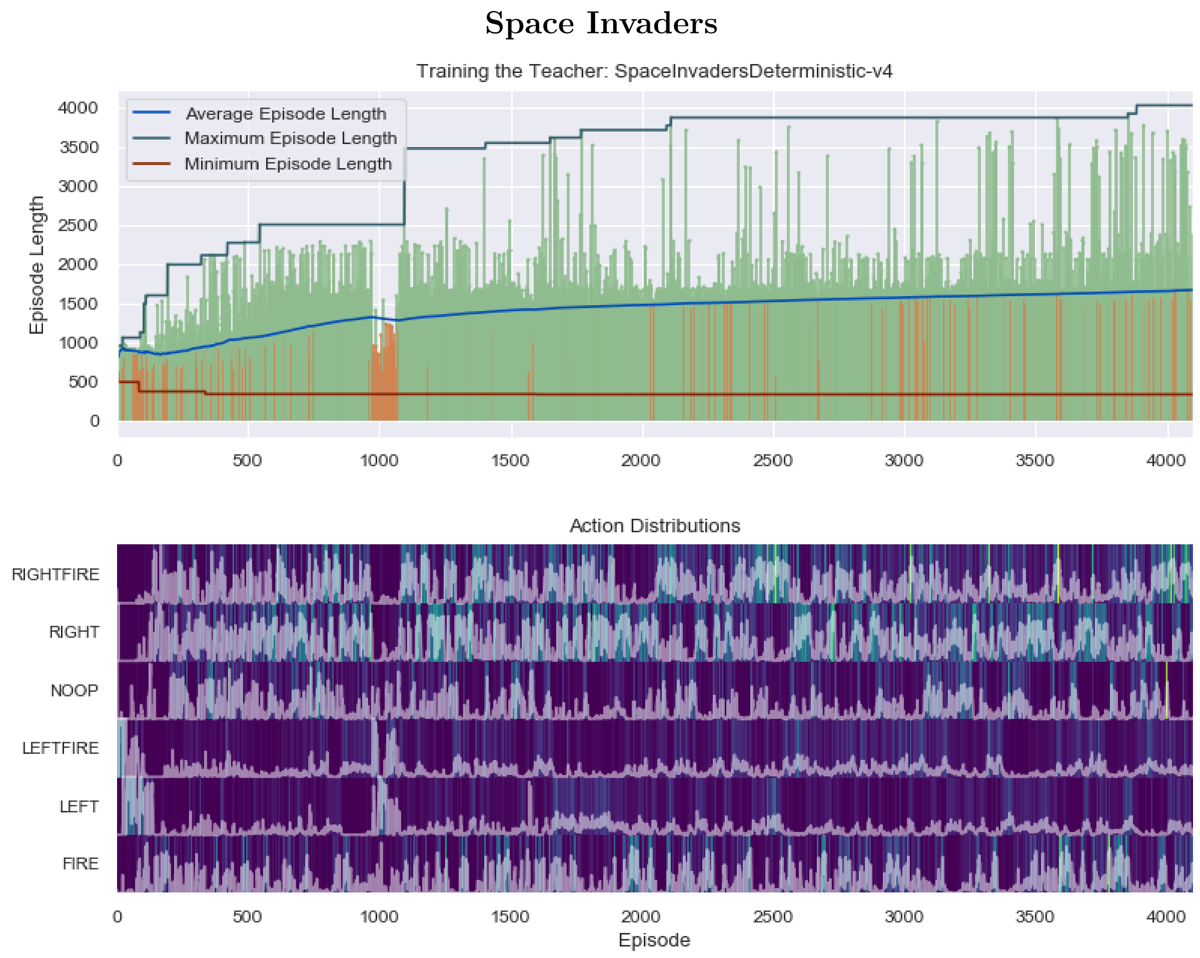


Figure 2.3.8: The performance of the Teacher model for Space Invaders. Data is plotted at every second training episode.

Figure 2.3.9 compares the actions of the Teacher and the baseline model for Space Invaders. As with Beamrider, the policy for the Teacher more often favours a certain action-type, whereas the baseline model better captures a balance across each episode.

To recap, we have seen how the reward signal crafted in section 2.2 can motivate a Teacher to learn an objective. The Teacher is trained *successfully* for Breakout and Beamrider, and *modestly* for Space Invaders. As attention now shifts towards the performance of the Student, we comment that the proficiency of the Teacher is expected to directly impact the quality of the Student's learning ability.

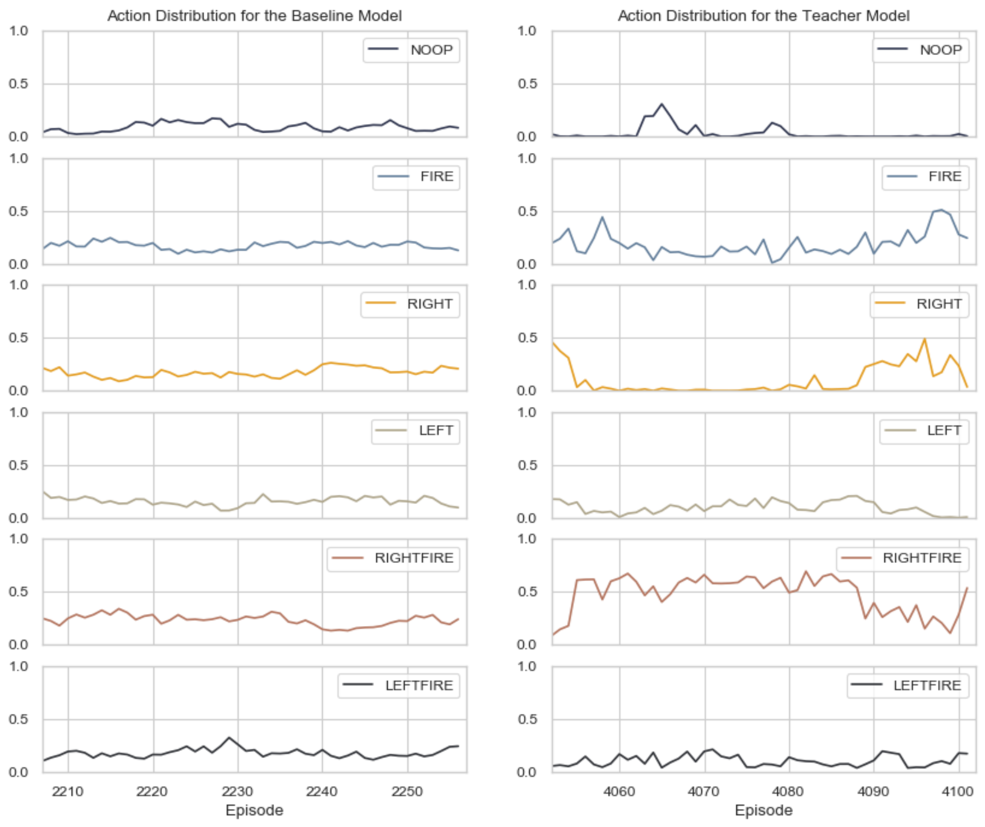


Figure 2.3.9: A comparative view of the action distributions between the baseline and Teacher models for Space Invaders. Data is displayed over the last fifty training episodes for each model.

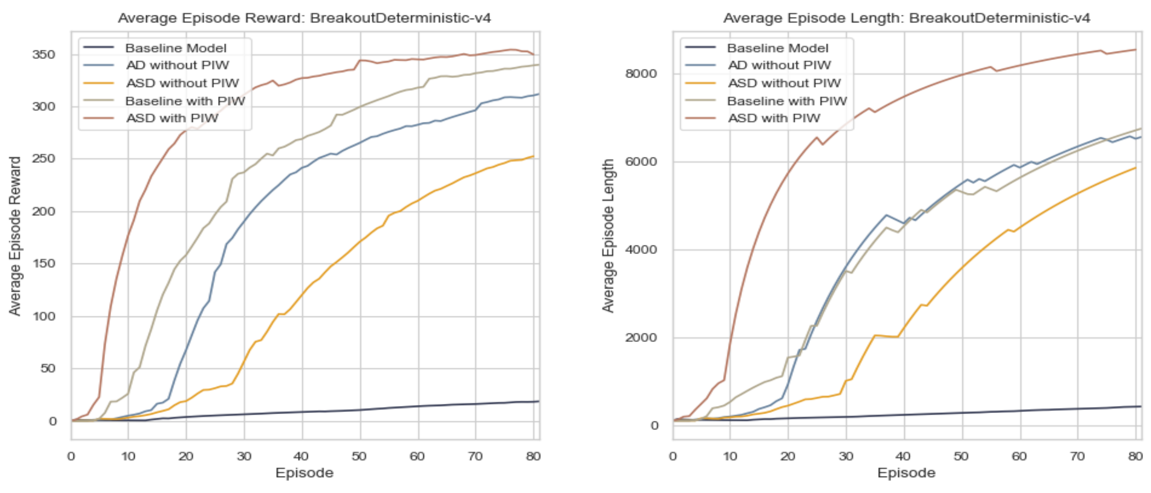
2.3.3 Student Performance

In this section, we present the learning curves for each Student model discussed in section 2.2. Where appropriate, we designate whether any preinitialized weights (PIW) have been prescribed to the Student from the Teacher. Performance summaries include a view of both the *average episode reward* and the *average episode length*, facilitating commentary as to the influence of the Teacher.

Beginning our analysis with Breakout. Figure 2.3.10 demonstrates that every Student model learned how to play Breakout faster than the baseline model. Given the simplicity of the game, and the proficiency of the Teacher, the results agree with our expectations.

The best performing Student model was the ASD learner with PIW, which converged after only 80 episodes to an average reward that exceeded what the baseline model achieved after 304 episodes (in reference to Figure 2.3.1). For further comparison, we include the learning curve for the, slightly-modified, baseline model when given PIW. The modification is done simply by adding a linear fully-connected layer after the LSTM cell. The fact that the ASD model outperforms the baseline model when both are given PIW provides lift to the claim that the Teacher/Student framework can demonstrate accelerated learning.

Breakout



(a) Breakout's Average Episode Reward. *(b) Breakout's Average Episode Length.*

Figure 2.3.10: A comparative performance summary for the Student models on Breakout.

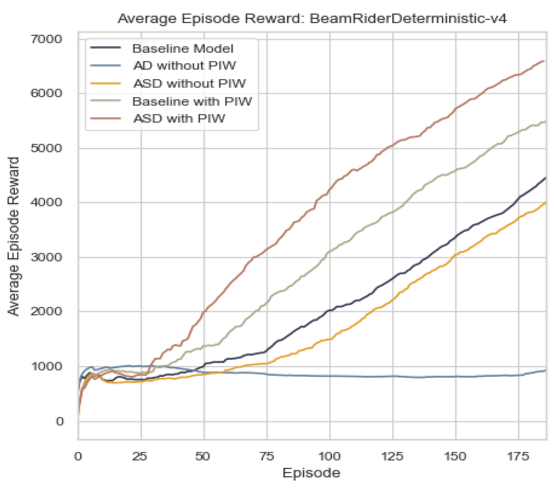
Notable is that the AD model outperformed the ASD model when neither were given PIW. While the ASD learner is better equipped for enriched learning of her task, the environment is just too simple in Breakout for this knowledge to culminate in accelerated learning. *Indeed, sometimes learning happens quickest when directly being informed of the answer.*

Plot (b) in Figure 2.3.10 shows that the Student models quickly adopted a survival-centric strategy. Given the similarities between the survival objective and the reward maximizing objective within the context of Breakout, plots (a) and (b) speak to the same conclusion: in view of rapid learning, the Teacher/Student framework was successful for Breakout.

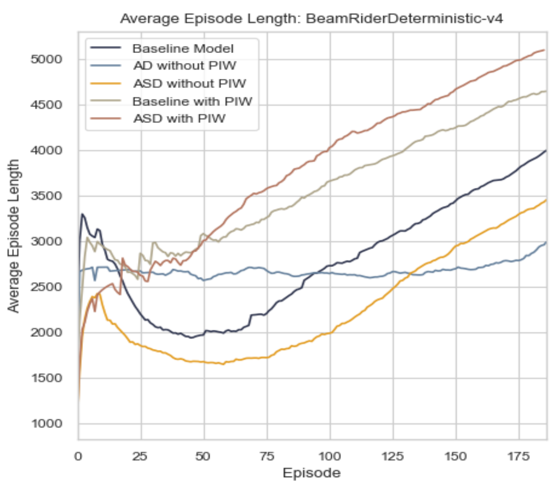
Progressing forward now to Beamrider. Figure 2.3.11 indicates that once again, the top performing model was the ASD learner with PIW. Differing from Breakout however, was the poor performance of the AD learner. Beamrider is a more complicated game than Breakout, whereby the survival objective from the Teacher appears to offer less relevance to the Student. We are led to conclude that an observation stream is *crucial* to the learning progression of the Student in a complicated environment.

The importance of PIW becomes clear, owing to the fact that the baseline model outperformed the ASD learner when PIW were not given. The interpretation is that PIW better enables the Student to understand, and act on, the advice coming from the Teacher. Viewed equivalently, in the absence of PIW, the Student needs to learn how to process advice from the Teacher *in addition to* learning the objectives required to excel at the task-at-hand. All of the justification is to say that PIW enables the Student to make enriched generalizations on her observation given the Teacher's advice.

Beamrider



(a) Beamrider's Average Episode Reward.



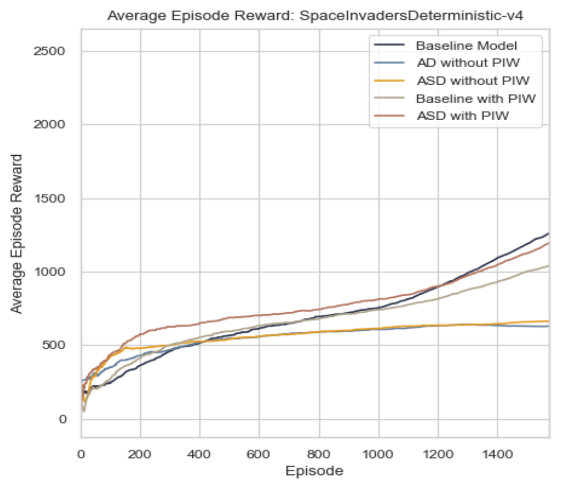
(b) Beamrider's Average Episode Length.

Figure 2.3.11: A comparative performance summary for the Student models on Beamrider.

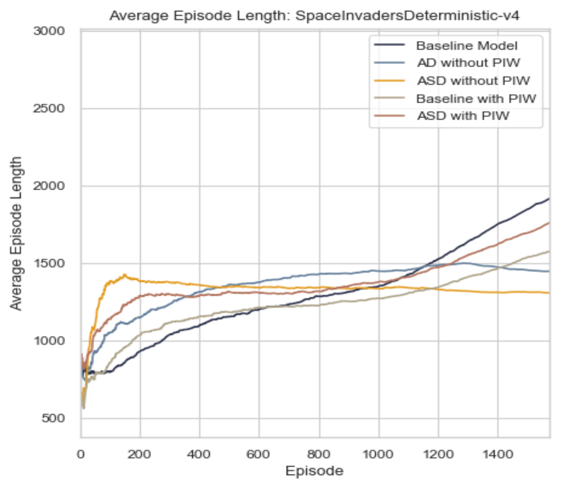
So far, it is clear that an observation stream with PIW has been the best approach for the Student in the case of Beamrider. Plot (b) from Figure 2.3.11 provides insight into why this is the case. The characteristic "dip" in the average episode length observed for both the baseline model and the ASD model without PIW is a pattern suggestive of a prolonged *exploration phase*. With little environmental context, the Student needs to spend more time gathering experience in order to begin generalizing the advice from the Teacher and reconciling observations from her state. On the other hand, the models prescribed with PIW are able to begin exploiting domain-specific knowledge sooner. Thus, *PIW accelerates learning by reducing the amount of exploration required.*

As previously observed, the Teacher models for Breakout and Beamrider learned to maximize survival, whereas the Teacher trained on Space Invaders did not manage survival to such the extent. We see how the suboptimality of the Teacher impacts the quality of the Student learning to play Space Invaders in Figure 2.3.12. Despite initially showing improvements over the baseline model, none of the Student learners outperformed at the end of the training session. The ASD learner with PIW appears to track the learning progression very closely, following a similar trajectory to the baseline model. The take-away here is that the success of a Teacher/Student learning framework depends not only on the relevance of the Teacher's advice, but also on the level of mastery achieved by the Teacher.

Space Invaders



(a) Space Invaders' Average Episode Reward.



(b) Space Invaders' Average Episode Length.

Figure 2.3.12: A comparative performance summary for the Student models on Space Invaders.

Interestingly, the baseline model performs better than its slightly modified version which is given PIW. This indicates that PIW prescription is not trivial for Transfer Learning models, and still relies on an adequate generalization process, for which the ASD learning model appears to demonstrate.

The results for the Student models are presented immediately below in Table 2.3.1; neatly shifting discussion towards future considerations.

Table 2.3.1: Summary of the performances of the Student learning models.

	Breakout			**Beamrider**			**Space Invaders**		
	Number of Episodes (A)	Average Reward	Average Reward after min(A)	Number of Episodes (B)	Average Reward	Average Reward after min(B)	Number of Episodes (C)	Average Reward	Average Reward after min(C)
▶ Baseline Model	304	202	18	278	6793	4418	**2256**	**2525**	**1258**
▶ Baseline with PIW	95	344	339	220	6103	5470	1701	1123	1042
▶ AD Model	107	338	310	264	1090	906	1572	628	628
▶ ASD Model	80	252	252	216	6326	3966	2215	732	662
▶ ASD Model with PIW	**80**	**350**	**350**	**185**	**6591**	**6591**	1806	1363	1192

2.4 Future Work

Through our analysis, we have demonstrated how a Teacher/Student learning framework can lead to accelerated learning. Our coverage has provided insight into the practical training schemes for both the Teacher and Student models.

In the case of the Teacher, we demonstrated how a crafted reward signal can be used to train a Deep Reinforcement Learning agent to learn a generalized task. The approaches were successful in two of the three applications. Such work is extendable outside the context of a Teacher/Student learning framework and offers relevance when designing a Deep Reinforcement Learning application from scratch.

From the perspective of the Student, we have shown two learning models that derive advice from a Teacher, extend decisioning to state observations and handle preinitialized weights. The success of the Student models, as measured by an accelerated rate of learning when compared to a baseline model, was achieved under different conditions in light of the environmental complexity, the relevance of the Teacher's advice, and the extent of the Teacher's own mastery of her advice.

Based on the results presented, we suggest two areas for further work.

▶ **Next-step 1:** The hyperparameters were tuned to the specifications from [36]. Despite noting strong performance with the baseline models, future work might materialize from the tuning of hyperparameters to match the specifics of each environment.

▶ **Next-step 2:** The Teacher trained under Equation 2.2.1 for Space Invaders did not achieve mastery in her survival, which impacted the ability of the ASD learner to best the baseline model in learning speed. Next-step considerations should focus on training the Teacher model to a level of mastery on Space Invaders. This would demonstrate the flexibility of reward signal engineering and can then be used to further validate the performance of the Student learning models.

Equation 2.2.1 was inspired by the unit-reward signal, as mentioned in section 2.2. In Figure 2.4.1, we show the learning curve for the Teacher trained under the unit-reward signal on Space Invaders. What becomes apparent is a quickly-realized entropy breakdown, where the Teacher settles into a suboptimal policy with little variability in her actions. This is perhaps a shortcoming specific to an n step ahead method, since any action performed by the agent along a given trajectory is rewarded the same.

While clearly the unit-reward signal "over-encourages" the Teacher, even Equation 2.2.1 seems to suffer from the poor-attribution and timing of reinforcement signalled to the Teacher. A proficient reward signal must therefore be one that offers a direct link between positive reinforcement and desired behaviour, as the causality of the linkage directly impacts learning progression.

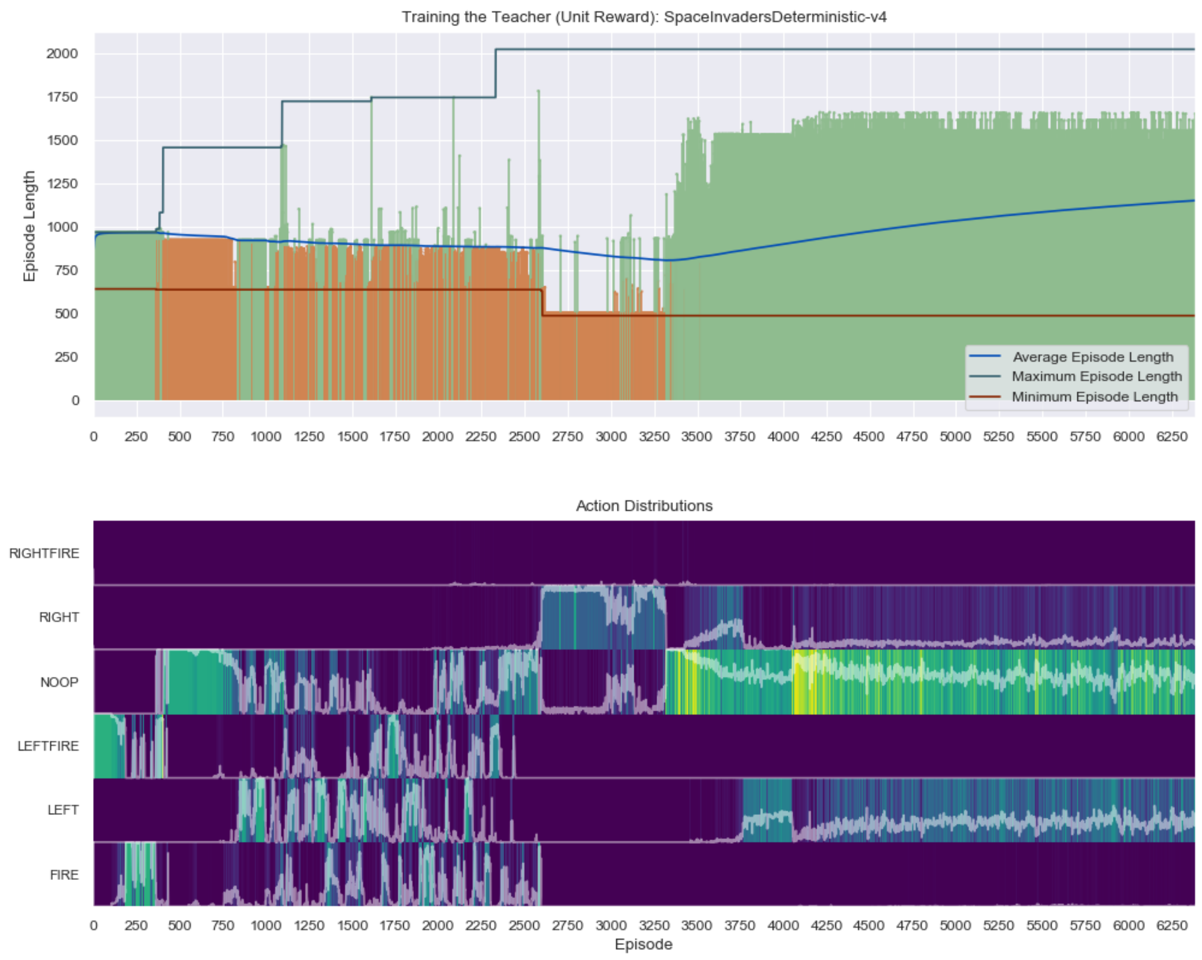


Figure 2.4.1: The performance of the Teacher model for Space Invaders trained under the unit-reward signal. Data is plotted at every second training episode.

Chapter 3

A Transient Price Impact Model for Order Execution

In this chapter, we study a models-driven approach to Order Execution. The model we introduce couples the dynamics for the bid- and ask-price processes, and is similar to that studied in [59]. Where our model differs from [59] is in the reversion of the bid- and ask-price processes to a fundamental, unaffected price process. Under the transient price impact model, we derive the trader's liquidation wealth and formulate the mean-variation maximization for an agent carrying constant absolute risk-aversion. In characterizing optimality, we are able to avoid a typical Hamilton-Jacobi-Bellman type equation and instead, show that the discretized model can be solved as a Quadratic Program. To the tune of [59], a supporting variational analysis is given to justify the discretization approach and formulate a continuous-time representation. The results admit insight into economic factors such as market depth, resilience, tightness and bias.

The roadmap for this chapter is as follows:

In section 3.1, we motivate the study of Order Execution by introducing the *Limit Order Book* and reviewing applications that will be relevant to our model.

In section 3.2, we introduce the transient price impact model and formulate the optimization objective.

In section 3.3, we begin by discretizing the objective and solving the resulting optimization problem as a Quadratic Program. Under certain market conditions, a closed-form solution can be derived. We make strides towards a continuous-time representation by evaluating the limit of the previously-found closed-form solution and show an agreement to [4]. In proceeding with further continuous-time analysis, we adopt a variational approach to confirm the results found for the analytical solution.

In section 3.4, we discuss extensions to the transient price impact model which includes trading multiple assets and competing against adversarial agents. The chapter is concluded on a note motivating the upcoming application of Deep Reinforcement Learning to Order Execution.

I am grateful for the guidance and support provided by Dr. Masaaki Fukasawa, under whose supervision this chapter's work was completed under.

3.1 Background

3.1.1 The Limit Order Book

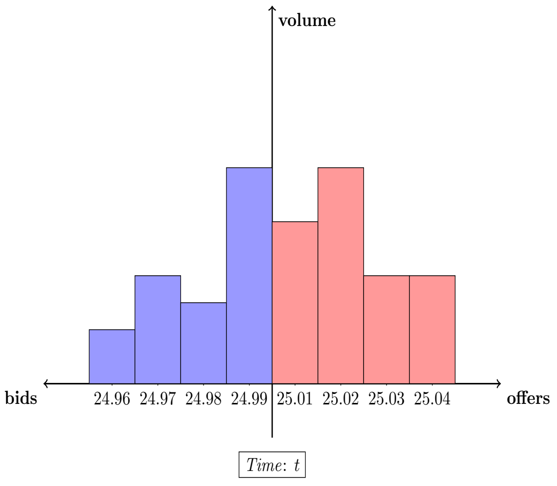
Securities trading has evolved significantly since the period of "Open-Outcry", where trading once took place through verbal and physical cues. This comes in staggering contrast to modern trading, which now occurs significantly over electronic markets. Electronic trading has its origins traced back to 1969, when the brokerage firm *Instinet* first offered a Direct Market Access (DMA) service to clients [67]. In the years since, Electronic Communication Networks (ECNs) became adopted by stock exchanges on a widespread basis and eventually, securities markets became digital. The Toronto Stock Exchange owns the claim of being the "first exchange in the world to computerize stock market trading" when the exchange became fully electronic in 1997[1].

Traders interact with electronic markets by posting *limit orders* to express the price and volume that they intend to trade at. A matching engine executes trades when the price of a bidding order, that coming from a buyer, crosses the price given by an offering/asking order, that coming from a seller. In a general sense, the execution of a trade is not guaranteed and naturally depends on the target-price of the posted limit order. When such a target-price is disregarded completely – indicating a trader's intention to immediately fill an order – then a *market order* is posted and is matched with the best bid or offer (BBO) price that is prevailing in the market. The *Centralized Limit Order Book* consolidates all such outstanding limit orders and, in the case that we will study, discloses this information to market participants.

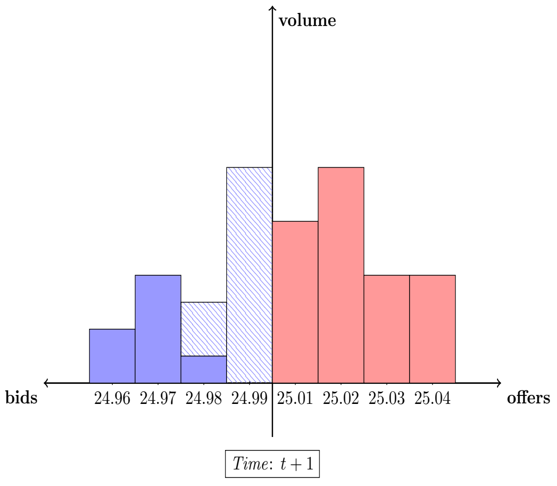
The limit order book plays host to a competitive and fast-paced world of trading, where spreads between the best bids and offers are usually kept tight, leading to lower transaction costs and enhanced liquidity [7]. A hypothetical limit order book is presented in Figure 3.1.1, which shows a market order "walking through" the limit order book. As it pertains to real data, a comprehensive empirical study of limit order books from exchanges in London, Paris, New York and Spain is given in [14].

Trading on a limit order book provides a glimpse into the complicated dynamics that drive the price of securities; the information content from the limit order book plays a role in the price discovery mechanism [17]. As a result, modelling the limit order book becomes central to the study of Market Microstructure [89]. There are many approaches to study the price dynamics in a limit order book, for instance, [48] constructs a Markov model by treating the arrival of limit orders as independent Poisson processes, each with a price that is informed from independent and identically distributed variables of

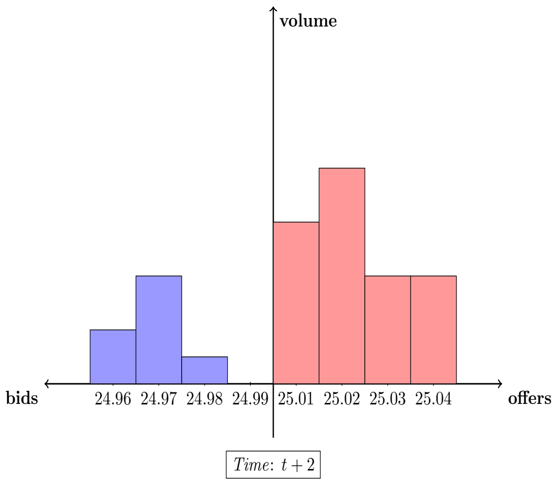
[1] https://www.tsx.com/trading



(a) The initial state of the limit order book.



(b) An active order depleting volume at the BBO.



(c) The new state of the limit order book.

Figure 3.1.1: A hypothetical limit order book with price evolution.

some density. Similarly, [73] works with a Markovian equilibrium model that arises when traders are permitted to cancel their limit orders. Outside of models with a Markov based-assumption, stochastic differential equations can be adapted to model quote-density [21] and various other diffusion models for prices. We appeal to [1] for a comprehensive overview of limit order book modelling.

In our approach, we will consider an extension of the block shaped limit order book from [4]. This decision is supported on account of simplicity and the intuitive capture of the underlying transient price dynamics. Under the block shaped limit order book, quote-density is uniformly distributed beyond the BBO and linear price impacts arise due to market orders. Furthermore, the bid- and ask-prices recover to a fundamental price process – one that is not impacted by trading volumes – at an exponential rate.

The block shaped limit order book has prompted several extensions, including that under generalized densities or "shapes" as in [69]. The block shaped limit order book has also been studied under high-resiliency limits in [47], showing a resemblance in the context of Order Execution to the seminal findings from [72], making the block shaped limit order book particularly appealing. The transient price impact model that we study is inspired by [59], which considers a two-sided block shaped limit order and defines a coupled system for the bid- and ask-price processes.

The limit order book is an entry point for many applications that hinge on price dynamics. We will now briefly review the application central to our work, that is *Optimal Order Execution*, but we acknowledge a more detailed investigation from [34].

3.1.2 Optimal Order Execution

A trader who is tasked with executing a large order must decide how, if at all, the quantity should be split over smaller trades. From Figure 3.1.1, it becomes clear that executing trades in large block-sizes leads to unfavourable execution prices and detrimental price impacts. However, if the trader posts limit orders that are too far from the BBO, then execution becomes unlikely. The tradeoff that these considerations pose is one between *price impacts* and *price uncertainties* [45]. The risk coming from price impact culminates in transactions costs when trading is executed too quickly, or in too large allotments, effectively stripping the market of liquidity and stipulating a penalty under the laws of supply and demand. In contrast, price uncertainty risk is detrimental for strategies which trade too slowly and expose the trader to uncertain market conditions. In dealing with the risk tradeoff, the consensus opinion is that a large order should be executed as several smaller ones [10][31]. The subject of Optimal Order Execution answers *just how many* trades should be executed and *just how small* these trades ought to be so as to maximize trading profits.

The first academic contribution to the study of Optimal Order Execution came from [12], where optimal trading policies were viewed as ones that minimize expected transaction costs. Execution prices were modelled as functions of the prevailing market price offset by a linear impact induced from the agent's trading quantities. Under the simplified approach, [12] demonstrated that a discrete share liquidation schedule can be formulated to realize an optimal execution strategy. The approach however, did not account for potential price fluctuations and as a result, served limited practical applications. Addressing this shortcoming, the seminal work of the "Almgren and Chriss model" [72] allowed for prices to be modelled as a Brownian Motion, and considered portfolio optimization from the view of minimizing the transaction costs which arise from both permanent and temporary market impacts. The Almgren and Chriss model advance linear cost functions and present the equivalent of an efficient frontier to characterize the class of optimal trading policies. Several direct extensions of the Almgren and Chriss model are discussed in [37].

The Almgren and Chriss model has had a profound influence on how transaction costs treated in the applications of Order Execution and related variants thereof. The work from [13] considers equilibrium returns in view of a quadratic cost premium; [30] studies perfect hedging under permanent market impacts; and [29] specifically reviews a conservative delta hedging strategy with transaction costs.

It is common for Optimal Order Execution problems to borrow from advanced techniques in Stochastic Control. This will not be the focus for our present modelling. Rather, we will show how a transient price impact model, one that imparts quadratic transaction costs, can be modelled as a Quadratic Program. The simple approach is perhaps the most significant contribution of our efforts, but suffers from interpretability limitations. To address these limitations, we will make strides towards adopting a variational argument similar to that presented in [59], which our model is well suited for.

3.2 Methodology

In this section, we propose a price impact model and formulate the resulting optimal liquidation problem. Subsequent analysis will follow in a continuous time setting leading to the eventual discretization.

3.2.1 Transient Price Dynamics

The setup begins by considering a trader who is tasked with liquidating shares of an asset over a fixed time horizon. Allow γ_t to represent the trader's inventory at time t, which is modelled

$$\gamma_t = \gamma_{0^-} + \gamma_t^+ - \gamma_t^-$$

Where γ_{0^-} gives the inventory of shares prior to trading and $\gamma_t^\pm$ provides the quantity of shares purchased $(+)$ or sold $(-)$. The purchasing and selling processes are taken as non-decreasing cádlág functions, which allows the inventory representation to capture both continuous trading and block/impulse trading. As an item of notation, $\gamma_{t^-}^\pm$ denotes the left limit of $\gamma_t^\pm$ and the magnitude of any block/impulse trade is given by $\Delta\gamma_t^\pm = \gamma_t^\pm - \gamma_{t^-}^\pm$. Also, we impose the constraint that after executing all trades by time τ, the trader has no remaining shares $\gamma_\tau = 0$. Finally, the trader commits to her trading policy at the beginning and does not modify the schedule intertemporally, which means that γ_t is deterministic.

The characterization of γ_t may be summarized by the set of admissible trading policies

$$\Gamma = \Big\{ (\gamma_t = \gamma_{0^-} + \gamma_t^+ - \gamma_t^-)_{0 \le t \le \tau} : \; \gamma_t^\pm \text{ non-decreasing cádlág,} \\ \gamma_{0^-}^\pm = 0, \;\; \gamma_\tau = 0 \Big\}$$

The fundamental price $(P)_{t\geq 0}$ of the asset is the unperturbed price, that which is realized in the absence of market activity. We take $(P)_{t\geq 0}$ as an arithmetic Brownian Motion with constant exogenously determined coefficients μ, σ and fix a filtered probability space $(\Omega, \mathcal{F}, \mathbb{P})$ satisfying all the conditions it ought to.

Now, introducing the price model, which takes the asset's bid- and ask-price as

$$\begin{aligned} dA_t &= dP_t + \lambda d\gamma_t^+ - \alpha(A_t - P_t)dt \\ dB_t &= dP_t - \lambda d\gamma_t^- - \alpha(B_t - P_t)dt \end{aligned} \tag{3.2.1}$$

With positive constants λ and α. The dynamics of the resulting limit order book closely follows the form introduced by [4], under which purchases appreciate the best available ask-price and sales erode the best available bid-price. The magnitude of the price impacts depend on the depth of the limit order book as specified by λ. To capture transience, the bid- and ask-prices revert to P_t at a resiliency rate α. A simple representation of the dynamics captured by Equation 3.2.1 is presented in Figure 3.2.1 below.

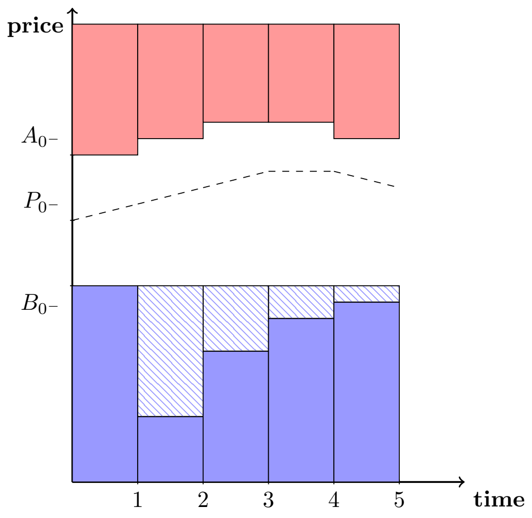


Figure 3.2.1: An example of the limit order book, as stipulated by Equation 3.2.1, reacting to a sell-order at $t = 2$.

As previously mentioned, Equation 3.2.1 is similar to the model considered in [59] which couples the bid- and ask-price processes through reversion to one-another. We suggest that reversion to P_t preserves the intuition behind the unperturbed price and better facilitates the dynamics of the asset's mid-quote $Q_t = (A_t + B_t)/2$. With the bid-ask spread $S_t = A_t - B_t$, Equation 3.2.1 gives rise to dynamics for Q_t and S_t

$$dQ_t = dP_t + \frac{1}{2}\lambda d\gamma_t - \alpha(Q_t - P_t)dt \tag{3.2.2}$$

$$dS_t = \lambda|d\gamma|_t - \alpha S_t dt \tag{3.2.3}$$

When taking for notation $|d\gamma|_t = d\gamma_t^+ + d\gamma_t^-$. It will be convenient to define a spread measuring the mid-quote bias $R_t = 2(Q_t - P_t)$, and under such, Equation 3.2.2 and Equation 3.2.3 are solved

$$R_t = R_{0^-} e^{-\alpha t} + \lambda \int_{[0,t]} e^{-\alpha(t-u)} d\gamma_u \tag{3.2.4}$$

$$S_t = S_{0^-} e^{-\alpha t} + \lambda \int_{[0,t]} e^{-\alpha(t-u)} |d\gamma|_u \tag{3.2.5}$$

A natural assumption on the initial conditions is that $|R_{0^-}| \leq S_{0^-}$, equivalent to imposing $B_{0^-} \leq P_{0^-} \leq A_{0^-}$.

3.2.2 Formulating the Optimization Problem

Now we address the trader's wealth process when executing an admissible policy. Under Equation 3.2.1, transaction costs are levied in proportion to the square of the block size of the trade. To see this, consider the wealth gained by the trader after selling $\Delta\gamma_t^-$ shares of the asset at $t \in [0, \tau]$

$$w_t = \int_0^{\Delta\gamma_t^-} (B_{t^-} - \lambda x) dx = B_{t^-} \Delta\gamma_t^- - \frac{\lambda}{2} |\Delta\gamma_t^-|^2$$

The wealth impact is analogous when the trader purchases, leading to the dynamics for the trading wealth

$$dw_t = \left(B_{t^-} - \lambda \Delta\gamma_t^-\right) d\gamma_t^- - \left(A_{t^-} + \lambda \Delta\gamma_t^+\right) d\gamma_t^+ \tag{3.2.6}$$

Such a relationship suggests that simultaneously purchasing and selling will always be suboptimal. It is therefore assumed, without loss of generality, that $\Delta\gamma_t^+ \Delta\gamma_t^- = 0$ for all $t \in [0, \tau]$.

The following proposition provides the liquidation wealth process, consistent with [77], which will inform our optimization objective.

Proposition 3.2.1 *For $\gamma \in \Gamma$, the liquidation wealth received at maturity is*

$$W_\tau(\gamma) = P_{0^-} \gamma_{0^-} + \int_0^\tau \gamma_{t^-} dP_t - \frac{1}{2} C_\tau(\gamma) \tag{3.2.7}$$

incurring transaction costs C_τ

$$\begin{aligned} C_\tau(\gamma) = R_{0^-} \int_{[0,\tau]} e^{-\alpha t} d\gamma_t + S_{0^-} \int_{[0,\tau]} e^{-\alpha t} |d\gamma|_t \\ + \lambda \left[\int_{[0,\tau]} \int_{[0,\tau]} e^{-\alpha|t-s|} d\gamma_s^- d\gamma_t^- + \int_{[0,\tau]} \int_{[0,\tau]} e^{-\alpha|t-s|} d\gamma_s^+ d\gamma_t^+ \right] \end{aligned} \tag{3.2.8}$$

PROOF: *Integrating Equation 3.2.6 over the trading horizon gives*

$$W_\tau(\gamma) = \int_{[0,\tau]} B_{t^-} d\gamma_t^- - \int_{[0,\tau]} A_{t^-} d\gamma_t^+ - \frac{\lambda}{2} \sum_{t\in[0,\tau]} |\Delta\gamma_t|^2$$

Which may be written in terms of R_t and S_t.

$$W_\tau(\gamma) = -\int_{[0,\tau]} P_{t^-} d\gamma_t - \frac{1}{2}\left[\int_{[0,\tau]} R_{t^-} d\gamma_t + \int_{[0,\tau]} S_{t^-} |d\gamma|_t + \lambda \sum_{t\in[0,\tau]} |\Delta\gamma_t|^2\right]$$

Integration by parts on $\int_{[0,\tau]} P_{t^-} d\gamma_t$ recovers the first two terms in Equation 3.2.7. The terms collected in the square brackets represents the transaction costs C_τ, which may then be written by substituting in Equation 3.2.4 and *Equation 3.2.5*

$$\begin{aligned} C_\tau(\gamma) = {} & R_{0^-} \int_{[0,\tau]} e^{-\alpha t} d\gamma_t + S_{0^-} \int_{[0,\tau]} e^{-\alpha t} |d\gamma|_t + \lambda \sum_{t\in[0,\tau]} |\Delta\gamma_t|^2 \\ & + \lambda\left[\int_{[0,\tau]} \int_{[0,t)} e^{-\alpha(t-s)} d\gamma_s^- d\gamma_t^- + \int_{[0,\tau]} \int_{[0,t)} e^{-\alpha(t-s)} d\gamma_s^+ d\gamma_t^+\right] \end{aligned}$$

Which is equivalent to Equation 3.2.8 under the respective integrand transformation. ■

We assume that the trader carries constant absolute risk-aversion, so the utility function is exponential with risk-aversion $\beta \geq 0$, $u(x) = e^{-\beta x}$. Seeing as the liquidation wealth is normally distributed, we obtain

$$\mathbb{E}\Big(u(W_\tau)\Big) = \exp\Big(-\beta\mathbb{E}(W_\tau) + \frac{\beta^2}{2}\mathrm{Var}(W_\tau)\Big)$$

Solving for the optimal trading policy amounts to maximizing the mean-variance objective function

$$J_\tau(\gamma) = P_{0^-}\gamma_{0^-} + \mu\int_0^\tau \gamma_{t^-} dt - \frac{1}{2}C_\tau(\gamma) - \frac{\beta\sigma^2}{2}\int_0^\tau \gamma_{t^-}^2 dt \to \max_{\gamma\in\Gamma}!$$

We carry this convex cost functional forward as we now address a solution method.

3.3 Main Results

In this section, we discretize the model and solve for the general cases computationally. We show several examples with varying market conditions and discuss the economic implications of the transient price impact model. Additionally, we work through a simple case which permits a closed-form solution under the method of Lagrangian multipliers. A limit of this closed-form solution shifts attention towards a continuous-time modelling approach, whereby a variational argument confirms our discrete-time approach.

3.3.1 Discrete-Time Optimization

Suppose we limit our focus to consider admissible trading policies that only execute at discrete points in time. It is convenient to evenly-space the time intervals between trades, in which case the discretization we take is over a grid $\Xi = \{0, h, 2h, \dots, \tau\}$ for $h = \tau/n$. A practical example is when a trader sells shares every fifteen minutes over single trading day, thereby executing $n = 26$ orders.

The set of admissible discrete policies is taken

$$\Gamma^n = \left\{ \left(\gamma^n_{ih} = \gamma_{0^-} + \sum_{j=0}^{i} (\Delta\gamma^+_{jh} - \Delta\gamma^-_{jh}) \right)_{ih\in\Xi} : \, (\Delta\gamma^\pm_{ih} \geq 0)_{ih\in\Xi}, \right. \\ \left. \gamma^n_\tau = 0 \right\}$$

and under discrete trading, the objective function is written

$$J_\tau(\gamma^n) = P_{0^-}\gamma_{0^-} - \mu h \sum_{i=0}^{n} (\Delta\gamma^+_{jh} - \Delta\gamma^-_{jh}) i - \frac{1}{2} C_\tau(\gamma^n) - U_\tau(\gamma^n) \tag{3.3.1}$$

with transaction costs C_τ

$$\begin{aligned} C_\tau(\gamma^n) = &\sum_{i=0}^{n} \Big[(S_{0^-} + R_{0^-}) \Delta\gamma^+_{ih} + (S_{0^-} - R_{0^-}) \Delta\gamma^-_{ih} \Big] e^{-\alpha ih} \\ &+ \lambda \sum_{i=0}^{n} \sum_{j=0}^{n} e^{-\alpha h|i-j|} (\Delta\gamma^+_{ih}\Delta\gamma^+_{jh} + \Delta\gamma^-_{ih}\Delta\gamma^-_{jh}) \end{aligned} \tag{3.3.2}$$

and risk charges U_τ

$$U_\tau(\gamma^n) = \frac{\beta\sigma^2 h}{2} \sum_{i=0}^{n} \left[\gamma_{0^-} + \sum_{j=0}^{(i-1)_+} (\Delta\gamma^+_{jh} - \Delta\gamma^-_{jh}) \right]^2 \tag{3.3.3}$$

Equation 3.3.2 induces two types of costs: the first comes from the initial market conditions stipulated by S_{0^-} and R_{0^-}, and the second comes from market impact. Such costs are linear and quadratic, respectively. Equation 3.3.3 returns a similar breakdown, and we show this below

$$\begin{aligned} U_\tau(\gamma^n) &= \frac{\beta\sigma^2 h}{2} \sum_{i=0}^{n} \left\{ \gamma^2_{0^-} + 2\gamma_{0^-} \sum_{j=0}^{(i-1)_+} (\Delta\gamma^+_{jh} - \Delta\gamma^-_{jh}) + \left[\sum_{j=0}^{(i-1)_+} (\Delta\gamma^+_{jh} - \Delta\gamma^-_{jh}) \right]^2 \right\} \\ &= \frac{\beta\sigma^2 h}{2} \sum_{i=0}^{n} \left\{ \gamma^2_{0^-} + 2\gamma_{0^-} \sum_{j=0}^{(i-1)_+} (\Delta\gamma^+_{jh} - \Delta\gamma^-_{jh}) + \sum_{j=0}^{(i-1)_+} (\Delta\gamma^+_{jh} - \Delta\gamma^-_{jh})^2 \right. \\ &\qquad \left. + \sum_{j=0}^{(i-1)_+} \sum_{k\neq j}^{(i-1)_+} (\Delta\gamma^+_{jh} - \Delta\gamma^-_{jh})(\Delta\gamma^+_{kh} - \Delta\gamma^-_{kh}) \right\} \end{aligned}$$

Risk charges therefore arise based on a planning horizon – the trader must weigh her decision to buy or sell shares at some time t against her decision to do so at every other instance in time. When $\beta > 0$ the trader is encouraged by risk considerations to liquidate her inventory more aggressively at the beginning of the trading horizon.

Equation 3.3.1 is quadratic in $\Delta\gamma^{\pm}$ and remains convex. As a result, we can efficiently model the solution using numerical software. We adopt the vector notation $\boldsymbol{\Delta\gamma}^{\pm} = \begin{bmatrix}\Delta\gamma_0^{\pm} & \cdots & \Delta\gamma_\tau^{\pm}\end{bmatrix}' \in \mathbb{R}^{n+1}$ and construct the Quadratic Program as follows

$$\begin{aligned} \underset{\boldsymbol{\Delta\gamma}^{\pm}}{\text{maximize}} \quad & P_{0^-}\gamma_{0^-} - \mu h \sum_{i=0}^{n} (\Delta\gamma_{jh}^{+} - \Delta\gamma_{jh}^{-}) i - \frac{1}{2} C_\tau(\gamma^n) - U_\tau(\gamma^n) \\ \text{subject to} \quad & \sum_{i=0}^{n} (\Delta\gamma_{ih}^{-} - \Delta\gamma_{ih}^{+}) = \gamma_{0^-}, \\ & \Delta\gamma_{ih}^{\pm} \geq 0 \quad \forall\ ih \in \Xi \end{aligned} \tag{3.3.4}$$

Which is equivalent to the standard-form representation,

$$\begin{aligned} \underset{\boldsymbol{\Delta\gamma}^{\pm}}{\text{minimize}} \quad & -\frac{1}{2}\begin{pmatrix}\boldsymbol{\Delta\gamma}^{+}\\ \boldsymbol{\Delta\gamma}^{-}\end{pmatrix}' \begin{pmatrix}-\lambda\boldsymbol{M}-\boldsymbol{V} & \boldsymbol{V}\\ \boldsymbol{V} & -\lambda\boldsymbol{M}-\boldsymbol{V}\end{pmatrix} \begin{pmatrix}\boldsymbol{\Delta\gamma}^{+}\\ \boldsymbol{\Delta\gamma}^{-}\end{pmatrix} \\ & -\begin{pmatrix}-(\boldsymbol{u}+\boldsymbol{c})-(S_{0^-}+R_{0^-})\boldsymbol{d}\\ \boldsymbol{u}+\boldsymbol{c}-(S_{0^-}-R_{0^-})\boldsymbol{d}\end{pmatrix}' \begin{pmatrix}\boldsymbol{\Delta\gamma}^{+}\\ \boldsymbol{\Delta\gamma}^{-}\end{pmatrix} \\ \text{subject to} \quad & \begin{pmatrix}-\boldsymbol{1}\\ \boldsymbol{1}\end{pmatrix}' \begin{pmatrix}\boldsymbol{\Delta\gamma}^{+}\\ \boldsymbol{\Delta\gamma}^{-}\end{pmatrix} = \gamma_{0^-}, \\ & \boldsymbol{I}\begin{pmatrix}\boldsymbol{\Delta\gamma}^{+}\\ \boldsymbol{\Delta\gamma}^{-}\end{pmatrix} \geq \boldsymbol{0} \end{aligned} \tag{3.3.5}$$

where $\boldsymbol{1} \in \mathbb{R}^{n+1}$ is a vector of ones; $\boldsymbol{M} \in \mathbb{R}^{(n+1)\times(n+1)}$ is the matrix

$$\boldsymbol{M} = \begin{pmatrix} 1 & m & m^2 & \dots & m^n \\ m & 1 & m & \dots & m^{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ m^n & m^{n-1} & m^{n-2} & \dots & 1 \end{pmatrix}$$

with $m = e^{-\alpha h}$; $\boldsymbol{V} \in \mathbb{R}^{(n+1)\times(n+1)}$ is the matrix

$$\boldsymbol{V} = \beta\sigma^2 h \begin{pmatrix} n & n-1 & n-2 & \dots & 1 & 0 \\ n-1 & n-1 & n-2 & \dots & 1 & 0 \\ n-2 & n-2 & n-2 & \dots & 1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix}$$

$\boldsymbol{u} \in \mathbb{R}^{n+1}$ is the vector

$$\boldsymbol{u} = \gamma_{0^-}\beta\sigma^2 h \begin{pmatrix} n & n-1 & n-2 & \dots & 1 & 0 \end{pmatrix}'$$

$\boldsymbol{c} \in \mathbb{R}^{n+1}$ is the vector

$$\boldsymbol{c} = \mu h \begin{pmatrix} 0 & 1 & 2 & \dots & n-1 & n \end{pmatrix}'$$

and $\boldsymbol{d} \in \mathbb{R}^{n+1}$ is the vector

$$\boldsymbol{d} = \frac{1}{2} \begin{pmatrix} 1 & m & m^2 & \dots & m^{n-1} & m^n \end{pmatrix}'$$

With the Quadratic Program in hand, we proceed to investigate the nature of the optimal trading policy under different market conditions.

3.3.2 Investigating the General Discrete Solution

In order to investigate the optimal trading policies in discrete-time, we construct a hypothetical market and distinguish exogenous variables between *agent-specific* parameters – those being under the agent's control – and *market-specific* parameters – those characterizing the nature of the market. The following parameter selections are taken for a baseline:

$$\begin{aligned} \textbf{Agent-specific}: &\quad \tau = 10; \quad \gamma_{0^-} = 10; \quad \beta = 1; \quad n = 200 \\ \textbf{Market-specific}: &\quad \lambda = 2; \quad \alpha = 1; \quad \mu = 5; \quad \sigma = 1; \quad S_{0^-} = 5; \quad R_{0^-} = 1 \end{aligned}$$

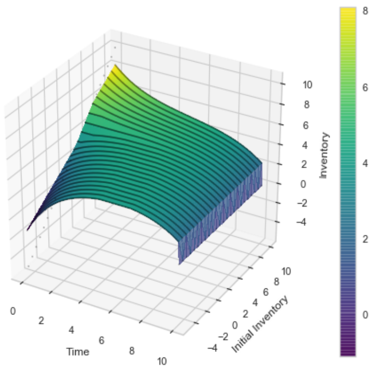
On an order of magnitude basis, the selection of the parameters are similar to that studied in [59].

Our approach will be to study how each parameter influences the optimal trading policy. Given the high-dimensional space, the analysis will be conducted on an "all else equal" assumption; the limitations of such an approach will directly motivate our efforts to adopt a continuous-time representation.

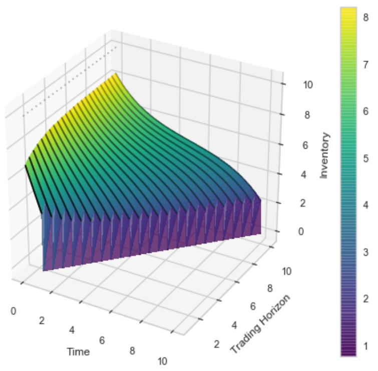
Figure 3.3.1 presents the findings when investigating the agent-specific parameters. We can tell that the agent is responding to a favourable μ estimate since trading is observed throughout the entire horizon. What's more, there are clearly periods of time when the optimal trading policy calls for purchasing, selling and waiting – consistent with the work from [59]. Several policies in Figure 3.3.1 depict large block trades made at the beginning and at the end of the trading horizon, suggesting an alignment to [4]. Indeed as we progress towards a closed-form solution, we show that the model we study preserves these findings.

From plot (a), we depict how the optimal policy is outlined for both liquidation and *acquisition* – in the case of the latter, the agent begins trading with $\gamma_{0^-} < 0$ and looks to *draw up* her position to close with $\gamma_\tau = 0$. The acquisition trajectories once again reflect favourable market conditions, where the agent is at times incentivized to acquire more shares than she intends to finish with, so as to realize the expected price appreciation.

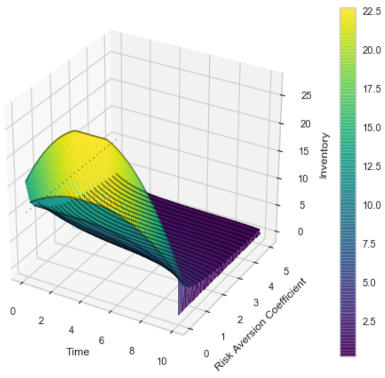
A similar, and far more aggressive strategy, is noticed in plot (c) when the agent trades with no risk-considerations, $\beta = 0$. The aggression however, quickly fades when the agent charges greater magnitude to her risk and instead, opts for large block trades at the beginning to quickly reduce her exposures.



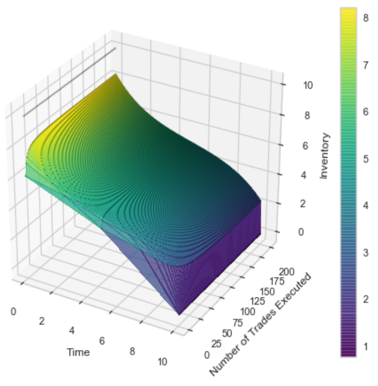
(a) Optimal policies under different γ_0-



(b) Optimal policies under different τ



(c) Optimal policies under different β tolerances



(d) Optimal policies under different n

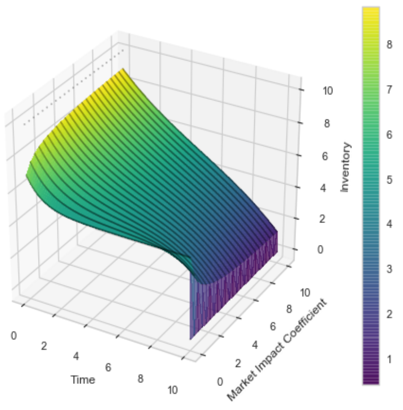
Figure 3.3.1: Optimal policies when agent-specific parameters are varied.

The main insight from plot (b) points to the idea that the time horizon primarily impacts the rate of trading. When an agent is able to trade over a longer period of time, it is advantageous from a market impact perspective to trade in smaller allotments. Whereas plots (a), (b) and (c) all show policies with fundamentally different approaches, plot (d) is relatively consistent and mute. We can take this as evidence indicating that a discrete-time policy converges point-wise towards a continuous-time policy.

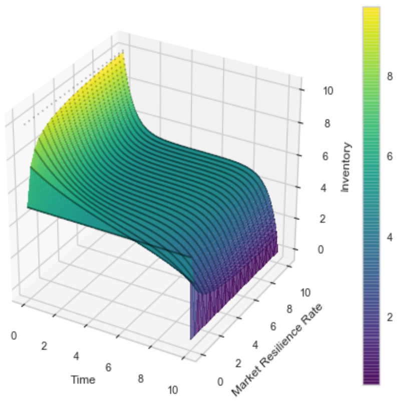
Studying different agent-specific parameters reveals how different investors would perform in the same market. However, a trader often does not have complete control over her execution horizon or in the number of shares that she needs to liquidate – Agency Trading, or trading on behalf of another institution, is an example where a trader is constrained in terms of both γ_{0-} and τ. Then, the natural question to pose is *how does the market influence the nature of the optimal trading policy?*

To advance discussions, Figure 3.3.2 shows how the optimal trading policy responds through a range of varying market conditions. When the market is governed by a strict impact coefficient, then liquidity becomes scarce, and the optimal policies respond by trading in smaller allotments. The opposite nature is observed when the resiliency rate is reduced; when the bid- and ask-prices recover to the fundamental price at slower rates, then optimal trading calls for larger block trades and a slower effective rate of trading.

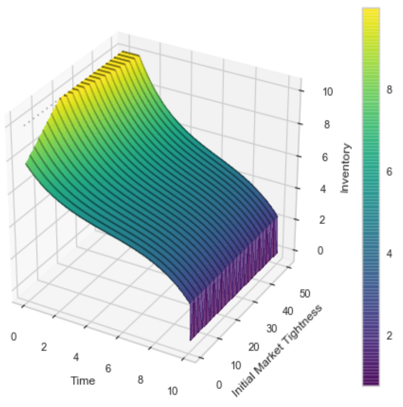
The initial state of the market, as measured through the spread-tightness and the mid-quote bias, appear to influence a holding period when conditions are unfavourable for liquidation. This would correspond to when the bid-ask spread is sufficiently large or when the fundamental price is sufficiently below the mid-quote. In either case, the optimal policies begin liquidating when the upfront costs are reduced simply from price appreciation.



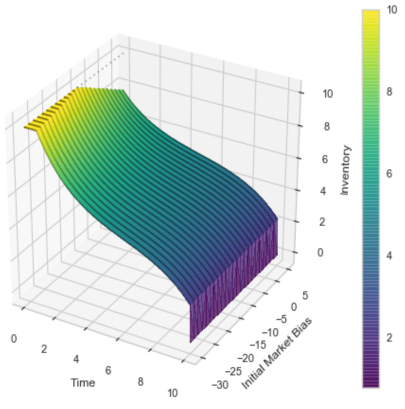
(a) Optimal policies under different λ coefficients



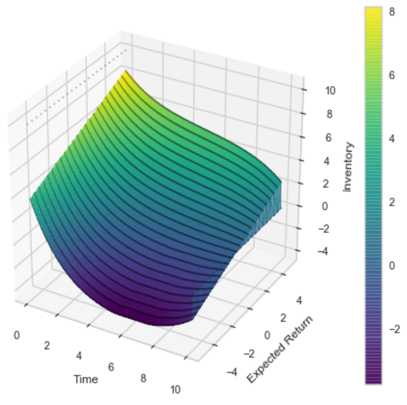
(b) Optimal policies under different α rates



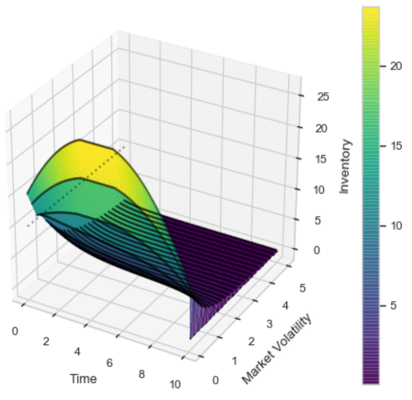
(c) Optimal policies under different S_{0-} values



(d) Optimal policies under different R_{0-} values



(e) *Optimal policies under different μ estimates*



(f) *Optimal policies under different σ estimates*

Figure 3.3.2: Optimal policies when market-specific parameters are varied.

Further generalizations are difficult to make in this discrete-time setting. Without a closed-form solution, it is challenging to infer a clear picture as to how each parameter changes the nature of the optimal trading policy. What remains however, is that the solution space is richly diverse and, at least at a surface level, offers insights that are intuitive from an economic perspective. We use this as motivation to carry forward in approaching a continuous-time representation of this problem.

3.3.3 A Closed-Form Discrete-Time Solution

While Equation 3.3.4 cannot be solved analytically under a general case, with simplifying assumptions that constitute a *simple market*, we do have an analytical solution for a risk neutral agent. The assumptions on the market are given below.

Definition 3.3.1 *A simple market is one where*

- *the fundamental price process is a martingale, so* $\mu = 0$*;*
- *there are no initial spreads,* $S_{0^-} = R_{0^-} = 0$*;*

Thus, optimal liquidation in a simple market amounts only to cost minimization. It becomes clear that purchasing should be ruled out entirely when $\gamma_{0^-} > 0$. This gives, without loss of generality, $\boldsymbol{\Delta\gamma}^+ \equiv 0$.

Remark 3.3.1 *If the agent is looking to optimally acquire shares, this would support the case whereby* $\gamma_{0^-} < 0$*, thus forcing* $\boldsymbol{\Delta\gamma}^- \equiv 0$*.*

Remark 3.3.2 *It is permissible for $\gamma_{0^-} = 0$ in this simple market setting, in which optimality clearly demands no trading activity.*

The solution under the simple market is presented in the following proposition.

Proposition 3.3.1 *For a risk-neutral agent seeking liquidation in the simple market, the optimal discrete trading policy $\boldsymbol{\Delta}\hat{\boldsymbol{\gamma}}^-$ is*

$$\boldsymbol{\Delta}\hat{\boldsymbol{\gamma}}^- = \frac{\gamma_{0^-}}{\mathbf{1}'\boldsymbol{M}^{-1}\mathbf{1}}\boldsymbol{M}^{-1}\mathbf{1} \tag{3.3.6}$$

Proof: *It may readily be shown that Equation 3.3.6 satisfies the necessary Karush-Kuhn-Tucker conditions for inequality-constrained optimization objectives. In doing so, one will observe that the positivity constraints are nonbinding, meaning that $\hat{\boldsymbol{\gamma}} > 0$ for the optimal solution. Consequently, the unconstrained optimization can be solved using Lagrangian multipliers on the following*

$$\mathcal{L}(\boldsymbol{\Delta\gamma}^-; l) = \frac{1}{2}(\boldsymbol{\Delta\gamma}^-)'\boldsymbol{M}\boldsymbol{\Delta\gamma}^- - l(\mathbf{1}'\boldsymbol{\Delta\gamma}^- - \gamma_{0^-})$$

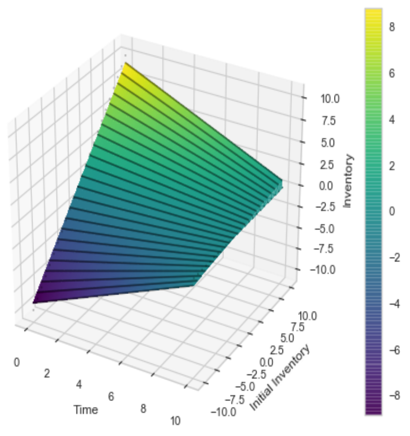
By construction, the matrix $\boldsymbol{M}$ is indeed invertible, with the inverse given as

$$\boldsymbol{M}^{-1} = \frac{1}{1-m^2}\begin{pmatrix} 1 & -m & 0 & \cdots & 0 & 0 & 0 \\ -m & 1+m^2 & -m & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -m & 1+m^2 & -m \\ 0 & 0 & 0 & \cdots & 0 & -m & 1 \end{pmatrix}$$

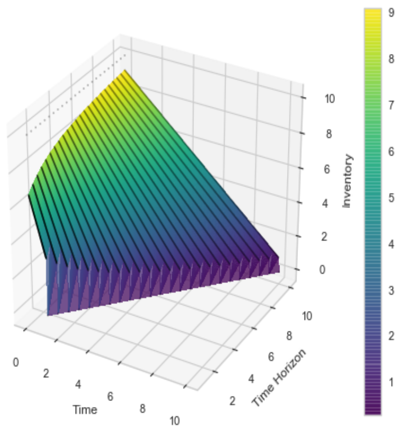
So completing the normal routine returns $\boldsymbol{\Delta}\hat{\boldsymbol{\gamma}}^-$ as per Equation 3.3.6. ■

From this, we can tell that the market impact plays *no role at all* in the optimal solution under a simple market. The only market-specific parameter that influences the optimal trading decision is the market resilience rate. Suggesting then that cost minimization calls directly for the timing of trades to match the rate at which the market recovers from any dislocation.

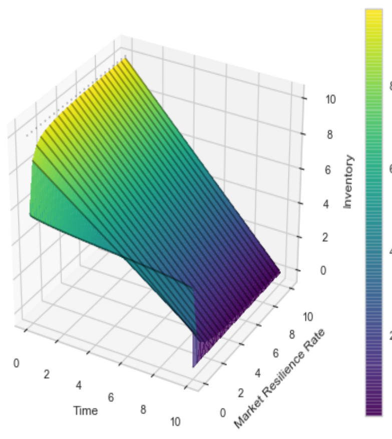
We then can view the plots in Figure 3.3.3 and notice that the optimal policies all appear to linear, which would suggest an alignment to [4]. This is understandable since the market setup, that with simplifying assumptions, is identical. We use this observation to bring us towards a continuous-time representation.



(a) *Simple optimal policies under different γ_{0^-}*



(b) *Simple optimal policies under different τ*



(c) *Simple optimal policies under different α rates*

Figure 3.3.3: Optimal policies in a simple market when varying agent-specific parameters

3.3.4 Towards a Continuous-Time Representation

We saw in Figure 3.3.1 that the optimal trading policy appears to quickly converge point-wise when the number of discrete trades gets large. We also noticed that the closed-form analytical solution for the discrete model demonstrates identical characteristics to that from [4]. It is very natural then to consider a limiting case for $n \to \infty$ and by doing so, we confirm that our results for the simple discrete model do in fact match with [4].

Proposition 3.3.2 *The continuous-time limit of Equation 3.3.6 is the function*

$$\gamma_t^- = \begin{cases} \frac{\gamma_{0^-}(1+\alpha t)}{2+\alpha\tau} & \text{if } t \in [0,\tau) \\ \gamma_{0^-} & \text{otherwise} \end{cases} \tag{3.3.7}$$

Proof: *Following directly from Equation 3.3.6, the following can be written*

$$\begin{aligned}\Delta\gamma_0^- = \Delta\gamma_\tau^- &= \frac{\gamma_{0^-}(1-m)}{1-m^2+n(1-m)^2}\\ \Delta\gamma_{ih}^- &= \Delta\gamma_0^-(1-m) \quad \forall\ ih \in \Xi \setminus \{0,\tau\}\end{aligned}$$

And then, the rate of continuous trade is calculated by taking, for any $ih \in \Xi \setminus \{0,\tau\}$

$$\lim_{n\to\infty} \frac{\Delta\gamma_{ih}^-}{\tau/n} = \lim_{n\to\infty} \frac{\gamma_{0^-}}{\tau\left(\frac{1+m}{n(1-m)}+1\right)} \overset{LH}{=} \frac{\alpha\gamma_{0^-}}{2+\alpha\tau}$$

A similar calculation is done to show $\lim_{n\to\infty}\Delta\gamma_0^- = \gamma_{0^-}/(2+\alpha\tau)$*, thus allowing the function to be constructed.* ■

By this point, it comes as no surprise that the continuous equations are linear – we figured as much from the nature of Figure 3.3.3. We also had the foresight to see that the market impact coefficient does not influence the optimal liquidation strategy in a simple market. However, what we can now answer is precisely how the market resilience rate influences trading. In the limit where $\alpha \to 0$, the agent is compelled to avoid permanent price dislocations by continuously trading, and instead evenly allocates block trades at the beginning and the end of the horizon. In the other limit where $\alpha \to \infty$, trading is done uniformly throughout the horizon at a rate matching γ_{0^-}/τ, with no block trades at all.

A succinct, closed-form solution is what motivates our next approach.

3.3.5 A Variational Approach

In the analysis to follow, the mean-variance minimization will be preserved, however, the objective is taken under a different functional form. In leading to the new objective statement, we consider an alternative expression for the transaction costs C_τ, as presented in the following proposition.

Proposition 3.3.3 *A feasible trading policy* $\gamma \in \Gamma$ *will incur transaction costs which may be written*

$$\begin{aligned}C_\tau(\gamma) = R_{0^-}\int_{[0,\tau]} e^{-\alpha t} d\gamma_t + S_{0^-}\int_{[0,\tau]} e^{-\alpha t}|d\gamma|_t\\ + \frac{1}{2\lambda}\left[(R_\tau - R_{0^-}e^{-\alpha\tau})^2 + (S_\tau - S_{0^-}e^{-\alpha\tau})^2\right]\\ + \frac{\alpha}{\lambda}\int_0^\tau \left[(R_{t^-} - R_{0^-}e^{-\alpha t})^2 + (S_{t^-} - S_{0^-}e^{-\alpha t})^2\right]dt\end{aligned} \tag{3.3.8}$$

Proof: *The arguments to follow are based on Lemma 2.1 from [59]. The underlying theory is supported by [44], Definition I.4.45, Proposition I.4.49 a), and Theorem I.4.52.*

The transaction costs C_τ may be written

$$C_\tau(\gamma) = \int_{[0,\tau]} R_{t^-} d\gamma_t + \int_{[0,\tau]} S_{t^-} |d\gamma|_t + \lambda \int_{[0,\tau]} d[\gamma,\gamma]_t \tag{3.3.9}$$

Using Equation 3.3.9 as a starting point, consider the first term involving R. Plugging in Equation 3.2.4 yields

$$\int_{[0,\tau]} R_{t^-} d\gamma_t = R_{0^-} \int_{[0,\tau]} e^{-\alpha t} d\gamma_t + \lambda \int_{[0,\tau]} \int_{[0,t)} e^{-\alpha(t-u)} d\gamma_u d\gamma_t$$

Defining $Z_t = \int_{[0,t]} e^{\alpha u} d\gamma_u$, it follows that

$$\begin{aligned}\int_{[0,\tau]} R_{t^-} d\gamma_t &= R_{0^-} \int_{[0,\tau]} e^{-\alpha t} d\gamma_t + \lambda \int_{[0,\tau]} e^{-\alpha t} Z_{t^-} d\gamma_t \\ &= R_{0^-} \int_{[0,\tau]} e^{-\alpha t} d\gamma_t + \lambda \int_{[0,\tau]} e^{-2\alpha t} Z_{t^-} dZ_t \\ &= R_{0^-} \int_{[0,\tau]} e^{-\alpha t} d\gamma_t + \frac{\lambda}{2} \int_{[0,\tau]} e^{-2\alpha t} dZ_t^2 - \frac{\lambda}{2} \int_{[0,\tau]} e^{-2\alpha t} d[Z,Z]_t\end{aligned}$$

Applying integration by parts on the second term results in

$$\begin{aligned}\int_{[0,\tau]} R_{t^-} d\gamma_t = R_{0^-} \int_{[0,\tau]} e^{-\alpha t} d\gamma_t + \frac{\lambda}{2}\Big(e^{-2\alpha\tau} Z_\tau^2 - Z_{0^-}^2\Big) \\ + \lambda\alpha \int_0^\tau e^{-2\alpha t} Z_{t^-}^2 dt - \frac{\lambda}{2} \int_{[0,\tau]} e^{-2\alpha t} d[Z,Z]_t\end{aligned}$$

Which is followed by the back-substitution of $Z_{t^-} = e^{\alpha t}(R_{t^-} - R_{0^-} e^{-\alpha t})/\lambda$

$$\begin{aligned}\int_{[0,\tau]} R_{t^-} d\gamma_t = R_{0^-} \int_{[0,\tau]} e^{-\alpha t} d\gamma_t + \frac{1}{2\lambda}(R_\tau - R_{0^-} e^{-\alpha\tau})^2 \\ + \frac{\alpha}{\lambda} \int_0^\tau (R_{t^-} - R_{0^-} e^{-\alpha t})^2 dt - \frac{\lambda}{2} \int_{[0,\tau]} d[\gamma,\gamma]_t\end{aligned}$$

An identical routine shows a similar result for the second term in C_τ involving S. This leads to the desired form for C_τ. ■

Then following immediately from Proposition 3.3.3, the optimal continuous-time trading policy $\gamma \in \Gamma$ is the following minimizer

$$\hat{\gamma} = \arg\inf_{\gamma \in \Gamma} \quad J_\tau(\gamma) \tag{3.3.10}$$

with the convex cost functional, defined separately for $\beta = 0$ and $\beta > 0$

$$J_\tau(\gamma) = \begin{cases} \frac{\beta\sigma^2}{2} \int_0^\tau (\gamma_{t^-} - \frac{\mu}{\beta\sigma^2})^2 dt + \frac{1}{2} C_\tau(\gamma) & \text{if } \beta > 0 \\ \mu \int_0^\tau \gamma_{t^-} dt + \frac{1}{2} C_\tau(\gamma) & \text{otherwise} \end{cases} \tag{3.3.11}$$

and $C_\tau(\gamma)$ is taken as per Equation 3.3.8.

Following the approach established by [59], the time now comes to determine the buying- and selling-subgradients for $J_\tau(\cdot)$ which will come to define the first order optimality conditions. To facilitate this, allow for the term-wise separation of $J_\tau(\cdot)$ into

$$J_\tau(\gamma) = D_\tau(\gamma) + L_\tau(\gamma) + Q_\tau(\gamma) \tag{3.3.12}$$

where

$$D_\tau(\gamma) = \begin{cases} \frac{\beta\sigma^2}{2}\int_0^\tau (\gamma_t - \frac{\mu}{\beta\sigma^2})^2 dt & \text{if } \beta > 0 \\ \mu \int_0^\tau \gamma_t dt & \text{otherwise} \end{cases} \tag{3.3.13}$$

$$L_\tau(\gamma) = \frac{1}{2}R_{0^-}\int_{[0,\tau]} e^{-\alpha t} d\gamma_t + \frac{1}{2}S_{0^-}\int_{[0,\tau]} e^{-\alpha t}|d\gamma|_t \tag{3.3.14}$$

$$\begin{aligned} Q_\tau(\gamma) = {} & \frac{1}{4\lambda}\Big[(R_\tau - R_{0^-}e^{-\alpha\tau})^2 + (S_\tau - S_{0^-}e^{-\alpha\tau})^2\Big] \\ & + \frac{\alpha}{2\lambda}\int_0^\tau \Big[(R_{t^-} - R_{0^-}e^{-\alpha t})^2 + (S_{t^-} - S_{0^-}e^{-\alpha t})^2\Big] dt \end{aligned} \tag{3.3.15}$$

The exercise in determining the buying- $(+)$ and selling- $(-)$ subgradients $\nabla_t^\pm J_\tau(\gamma)$ is handled by the next proposition.

Proposition 3.3.4 *For the convex cost functional as per Equation 3.3.12, for all $t \in [0,\tau]$*

$$\nabla_t^\pm J_\tau(\gamma) = \nabla_t^\pm D_\tau(\gamma) + \nabla_t^\pm L_\tau(\gamma) + \nabla_t^\pm Q_\tau(\gamma) \tag{3.3.16}$$

where

$$\nabla_t^\pm D_\tau(\gamma) = \begin{cases} \pm\beta\sigma^2 \int_t^\tau (\gamma_u - \frac{\mu}{\beta\sigma^2}) du & \textit{if } \beta > 0 \\ \mu & \textit{otherwise} \end{cases} \tag{3.3.17}$$

$$\nabla_t^\pm L_\tau(\gamma) = \frac{1}{2}(S_{0^-} \pm R_{0^-})e^{-\alpha t} \tag{3.3.18}$$

$$\begin{aligned} \nabla_t^\pm Q_\tau(\gamma) = {} & \frac{1}{2}e^{-\alpha(\tau - t)}\Big[(S_\tau \pm R_\tau) - (S_{0^-} \pm R_{0^-})e^{-\alpha\tau}\Big] \\ & + \alpha\int_t^\tau \Big[(S_u \pm R_u) - (S_{0^-} \pm R_{0^-})e^{-\alpha u}\Big] e^{-\alpha(u-t)} du \end{aligned} \tag{3.3.19}$$

PROOF: *This proof is divided into two parts; firstly, the buying- and selling-subgradients will be calculated for D_τ (L_τ and Q_τ are analogous but slightly more tedious) and secondly, using the well known property of convex functions, $\nabla_t^\pm J_\tau(\gamma)$ will be calculated. The arguments here follow that of [59], Lemma 4.5.*

Part I: Calculating the subgradients. *Considering two trading policies $\xi, \gamma \in \Gamma$, each with the same starting inventories $\xi_{0^-} = \gamma_{0^-}$. Take $\epsilon \in (0,1]$ and define*

$$\delta_X = \lim_{\epsilon \to 0} \frac{X_\tau\Big(\epsilon\xi + (1-\epsilon)\gamma\Big) - X_\tau(\gamma)}{\epsilon}$$

The calculation for D_τ when $\beta > 0$ will be provided, with the remaining subgradients following suit analogously. Preliminary simplifications give

$$\delta_D = \beta\sigma^2 \int_0^\tau \Big(\gamma_t - \frac{\mu}{\beta\sigma^2}\Big)(\xi_t - \gamma_t)dt$$

Using $\xi_t - \gamma_t = \int_{[0,t]}(d\xi_u^+ - d\gamma_u^+) - \int_{[0,t]}(d\xi_u^- - d\gamma_u^-)$

$$\begin{aligned}\delta_D &= \beta\sigma^2 \int_0^\tau \int_{[0,t]} \Big(\gamma_t - \frac{\mu}{\beta\sigma^2}\Big)(d\xi_u^+ - d\gamma_u^+)dt \\ &+ \beta\sigma^2 \int_0^\tau \int_{[0,t]} \Big(\frac{\mu}{\beta\sigma^2} - \gamma_t\Big)(d\xi_u^- - d\gamma_u^-)dt\end{aligned}$$

An application of Fubini's Theorem yields the following

$$\begin{aligned}\delta_D &= \beta\sigma^2 \int_{[0,\tau]} \int_u^\tau \Big(\gamma_t - \frac{\mu}{\beta\sigma^2}\Big)dt\ (d\xi_u^+ - d\gamma_u^+) \\ &+ \beta\sigma^2 \int_{[0,\tau]} \int_u^\tau \Big(\frac{\mu}{\beta\sigma^2} - \gamma_t\Big)dt\ (d\xi_u^- - d\gamma_u^-)\end{aligned}$$

Which gives way for the assignment of $\nabla_t^\pm D_\tau(\gamma)$, provided that $\beta > 0$

$$\nabla_t^\pm D_\tau(\gamma) = \pm\beta\sigma^2 \int_t^\tau \Big(\gamma_u - \frac{\mu}{\beta\sigma^2}\Big)du$$

Part II: Convex Argument for the Construction of $\nabla_t^\pm J_\tau(\gamma)$.

$J_\tau(\gamma)$ is a convex function, so it holds that

$$\begin{aligned}&\epsilon J_\tau(\xi) + (1-\epsilon)J_\tau(\gamma) \geq J_\tau\Big(\epsilon\xi + (1-\epsilon)\gamma\Big) \\ \implies\ & J_\tau(\xi) + J_\tau(\gamma) \geq \lim_{\epsilon\to 0} \frac{J_\tau\Big(\epsilon\xi + (1-\epsilon)\gamma\Big) - J_\tau(\gamma)}{\epsilon} \\ &\geq \int_{[0,\tau]} \Big(\nabla_t^+ D_\tau(\gamma) + \nabla_t^+ L_\tau(\gamma) + \nabla_t^+ Q_\tau(\gamma)\Big)(d\xi_u^+ - d\gamma_u^+) \\ &\quad + \int_{[0,\tau]} \Big(\nabla_t^- D_\tau(\gamma) + \nabla_t^- L_\tau(\gamma) + \nabla_t^- Q_\tau(\gamma)\Big)(d\xi_u^- - d\gamma_u^-)\end{aligned}$$

Which leads to the desired assignment

$$\nabla_t^\pm J_\tau(\gamma) = \nabla_t^\pm D_\tau(\gamma) + \nabla_t^\pm L_\tau(\gamma) + \nabla_t^\pm Q_\tau(\gamma) \tag{3.3.20}$$

■

Having formulated the buy- and selling-subgradients, we can now proceed directly to a solution method.

3.3.6 Revisiting the Simple Market

As per [59], the first order optimality condition is the requirement that $\nabla_t^{\pm} J_\tau(\gamma) \geq 0$ over the trading horizon. Characterizing the subgradients over the entire state-space is challenging, and requires a dutiful parameterization over disjoint regions where the policy purchases, sells and holds. For such an approach, we defer to [59]. In proceeding, we note that the task greatly simplifies for trading policies which are absolutely continuous over the entire trading horizon, then $\nabla_t^{\pm} J_\tau(\gamma) = 0$. Recognizing that Proposition 3.3.2 gives a solution that is absolutely continuous over $(0, \tau)$, we now show how the variational argument we just constructed can be used to verify the continuous solution found in Proposition 3.3.2.

Formally stated, the proceeding objective is to use the first-order optimality condition to determine a continuous trading policy active over $(0, \tau)$ for a risk-neutral agent in a simplified setting. Recall that this means that the simple market carries no drift and introduces no initial tightness nor bias. These assumptions greatly simplify the expression of the subgradient, leading to $\nabla_t^{\pm} D_\tau(\gamma) = \nabla_t^{\pm} L_\tau(\gamma) = 0$. This gives the following to deduce the dynamics of the optimal policy

$$\nabla_t^{\pm} Q_\tau(\gamma) = \frac{1}{2} e^{-\alpha(\tau - t)}(S_\tau \pm R_\tau) + \alpha \int_t^\tau (S_u \pm R_u) e^{-\alpha(u-t)} du \tag{3.3.21}$$

Setting $\nabla_t^{-} Q_\tau(\gamma)$ equal to zero and taking the derivative of with respect to t gives

$$\frac{\alpha}{2} e^{-\alpha(\tau - t)}(S_\tau - R_\tau) + \alpha^2 \int_t^\tau (S_u - R_u) e^{-\alpha(u-t)} du - \alpha(S_t - R_t) = 0 \tag{3.3.22}$$

Now, multiplying by $e^{-\alpha t}$ and integrating by parts results in

$$\alpha \int_{[t,\tau]} e^{-\alpha u}(dS_u - dR_u) - \frac{\alpha}{2}(S_\tau - R_\tau) e^{-\alpha \tau} = 0 \tag{3.3.23}$$

Further differentiating over $(0, \tau)$ and once again multiplying by $e^{\alpha t}$ yields

$$dS_t - dR_t = 0 \tag{3.3.24}$$

By substituting in the dynamics from Equation 3.2.4 and Equation 3.2.5, the nature of $\dot{\gamma}_t^{-}$ is revealed to be

$$2\lambda d\gamma_t^{-} - \alpha(S_t - R_t)dt = 0 \tag{3.3.25}$$

$$\dot{\gamma}_t^{-} = \frac{\alpha}{2\lambda}(S_t - R_t) \tag{3.3.26}$$

Once more differentiating and further substituting the spread dynamics from Equation 3.2.2 and Equation 3.2.3 produces a second-order ODE for γ_t^{-}

$$d\dot{\gamma}_t^{-} = \frac{\alpha}{2\lambda}(dS_t - dR_t) \tag{3.3.27}$$

$$d\dot{\gamma}_t^{-} = \frac{\alpha}{2\lambda}\Big[2\lambda d\gamma_t^{-} - \alpha(S_t - R_t)dt\Big] \tag{3.3.28}$$

$$\ddot{\gamma}_t^{-} = 0 \tag{3.3.29}$$

Which clearly implies a linear form for the continuous trading policy, which we write as

$$\gamma_t^- = ct + d \tag{3.3.30}$$

for $c, d \in \mathbb{R}$. What remains to be determined are the two boundary conditions that would specify the solution uniquely. We already know that the optimal trading policy calls for a block trade at the beginning and at the end of the trading horizon. We use this prior understanding to extend our framework to allow for such – in fact, we even know that the magnitudes of these block trades are equal, but we will show that this result follows naturally.

Consider the block trade made at the beginning, of size $\Delta\gamma_0^-$. Then $d = \Delta\gamma_0^-$ follows immediately. Now after this block trade is made, both of the induced spreads react such that $R_0 = -\lambda\Delta\gamma_0^-$ and $S_0 = \lambda\Delta\gamma_0^-$. This then directly provides inference to $c = \alpha\Delta\gamma_0^-$, which comes from differentiating Equation 3.3.30 and equating it to $\dot{\gamma_0}^- = \frac{\alpha}{2\lambda}(S_0 - R_0)$ from Equation 3.3.26.

To summarize, our reasoning thus far has returned $\gamma_t^- = \Delta\gamma_0^-(1+\alpha t)$ for all $t \in (0, \tau)$. This form indeed matches that of Proposition 3.3.2, but what still needs to be determined is the magnitude of $\Delta\gamma_0^-$ and $\Delta\gamma_\tau^-$. To this effect, we appeal to the fact that a feasible strategy is one that completely liquidates the inventory holdings, so $\int_{[0,\tau]} d\gamma_t^- = \gamma_0^-$. Evaluating this integral results in an expression which relates the two block trades $\Delta\gamma_0^-$ and $\Delta\gamma_\tau^-$

$$\Delta\gamma_0^-(1 + \alpha\tau) + \Delta\gamma_\tau^- = \gamma_0^- \tag{3.3.31}$$

The final step is to then recognize that Equation 3.3.29 indicates that Equation 3.3.26 is constant. As a result, evaluating Equation 3.3.23 with $t = \tau$ directly reveals that $\Delta\gamma_0^- = \Delta\gamma_\tau^-$. Finally, Equation 3.3.31 gives $\Delta\gamma_0^- = \gamma_0^-/(2+\alpha\tau)$ to finish our case study.

Indeed, we can proceed further with the variational arguments to recover dynamics for a more generalized continuous-time system. However, we acknowledge that these efforts would closely follow that from [59], and conclude on the verification of the results we first derived under the simple market.

3.4 Future Work

As a summary, we have taken a transient price impact model and have shown how optimal liquidation can be approached on the induced price dynamics. The main contribution from our work is with the discrete-time modelling of the optimization objective as a Quadratic Program. This very simple technique comes in direct contrast to the more complicated, and often intractable, technique of using a Hamilton-Jacobi-Bellman style equation. The significance of our work is demonstrated from the alignment of our results to that from inspiring sources in [4] and [59].

We acknowledge two areas of continued work that inform next-step considerations:

▶ **Next-step 1:** We credit the variational approach from [59] as a strong method to recover dynamics of optimal liquidation strategies under a transient price impact model. Continued work would pick-up directly where we leave off, and expand the variational argument outside of the simple market assumptions. Through preliminary work, we propose that the dynamics of the optimal liquidation policy γ_t^- abides by

$$\ddot{\gamma_t}^- = \frac{\alpha^2\beta\sigma^2}{2\alpha\lambda + \beta\sigma^2}\left(\frac{\mu}{\beta\sigma^2} - \gamma_t\right)$$

With the general form

$$\gamma_t^- = c_1 e^{\theta t} + c_2 e^{-\theta t} + \gamma_0 - \frac{\mu}{\beta\sigma^2}$$

Matching exactly that from [59]. Future work should prove this proposed dynamic.

▶ **Next-step 2:** A comprehensive extension of our work is to expand the dimensionality. Whereas we focus only on liquidating a single asset, if $(A)_{t\geq 0}$, $(B)_{t\geq 0}$ and $(P)_{t\geq 0}$ become p-dimensional then market-parameters are written as the matrices $\Lambda \in \mathbb{R}^{p\times p}$ and $\alpha \in \mathbb{R}^{p\times p}$. Furthermore, if adversarial agents are introduced through an aggregate inventory process ξ, then a Nash equilibrium can be studied.

These proposed modifications would culminate in price dynamics given by

$$\begin{aligned} dA_t &= dP_t + \Lambda(d\gamma_t^+ + d\xi_t^+) - \alpha(A_t - P_t)dt \\ dB_t &= dP_t - \Lambda(d\gamma_t^- + d\xi_t^-) - \alpha(B_t - P_t)dt \end{aligned}$$

A further extension of work along the vein of Order Execution is to consider a models-free approach. For this, we proceed to the next chapter to outline an application of Deep Reinforcement Learning for Order Execution.

Chapter 4

An A3C Model for Order Execution

For real securities, price dynamics are more complicated than that studied in chapter 3. Limit order books are generally not block shaped [68]; market impacts are not linear [97] [11]; market resiliency is a function of aggregated trading activity on the exchange [49]; and external events play a role in price discovery as well. Modelling for all such factors greatly encumbers analytical approaches and quickly leads to intractable systems. Reinforcement Learning offers an appealing models-free substitute to make abstract these complicated dynamics.

The roadmap for this chapter is presented as follows:

In section 4.1, we broadly introduce various Machine Learning techniques applied to financial data and formalize the Order Execution objective as it relates to Volume-Weighted Average Price (VWAP) execution.

In section 4.2, we present an approach to develop a Deep Reinforcement Learning model to tackle the Order Execution objective. As in chapter 2, we utilize an A3C algorithm and attempt to teach an autonomous agent to perform VWAP execution. We develop three candidate environments for simulated trading; the first environment derives from the block shaped limit order book, the second uses historical data from Nasdaq, and the third draws from a real-time market simulator. We progress our application on the last environment, but realize only modest results in section 4.3.

Recognizing ample areas of improvement, section 4.4 concludes the work from this chapter on a proposed connected to the Advice & State-Driven (ASD) decisioning model studied in chapter 2.

4.1 Background

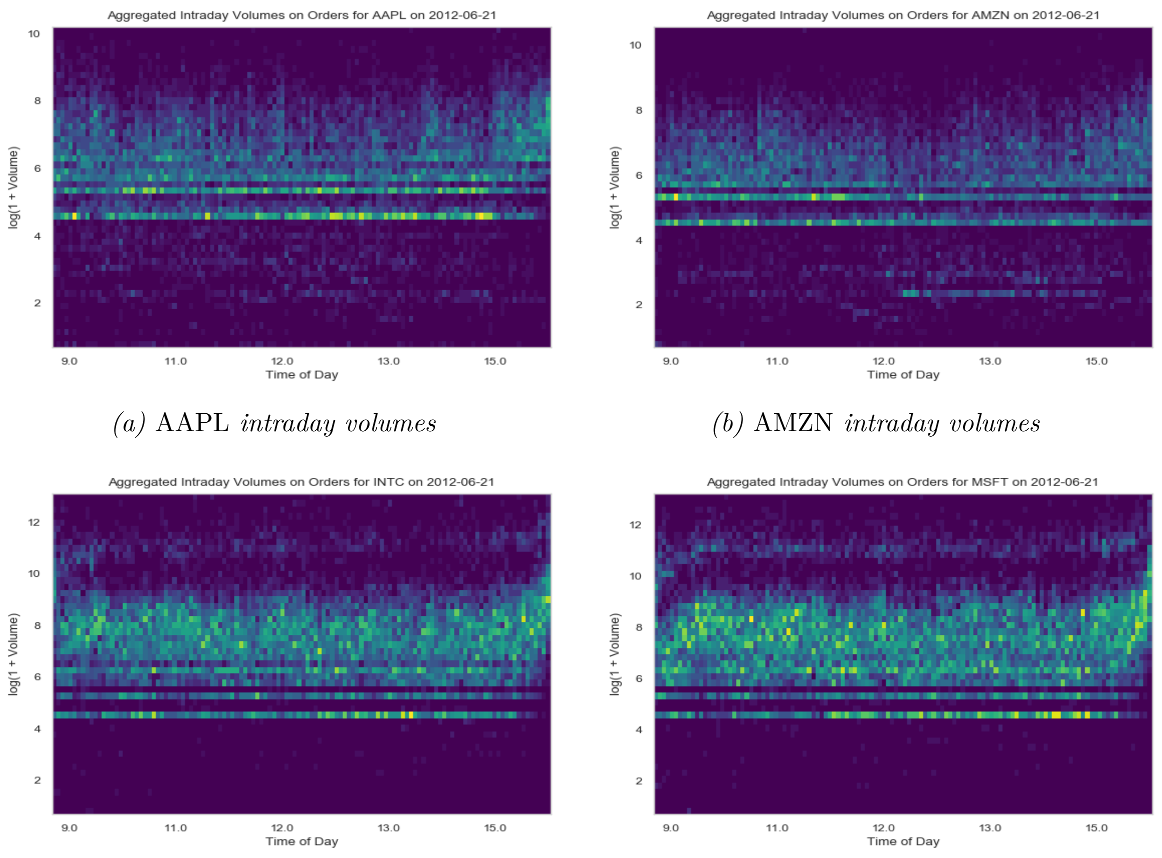
In chapter 3, we saw how a risk-informed agent would maximize her profits under a transient price impact model. Many practical applications of Order Execution tie trading to a benchmark, such as VWAP or the Time-Weighted Average Price (TWAP) [53].

Considering a benchmark execution strategy paves the way for Transaction Cost Analysis (TCA) to indicate ex-post how a trader's execution compared against the average market execution over the same time horizon. In this vein, execution that bests VWAP is viewed favourably and is crucial for winning client-flow in a competitive landscape.

The content in this section introduces VWAP execution strategies and their related applications with Reinforcement Learning. General discussions pertaining to feature-normalization for use-cases with financial data is also given and becomes relevant as we discuss our modelling approach in section 4.2.

4.1.1 VWAP Trading Strategies

It is generally accepted that more trading activity is observed near market-open and market-close, characterizing a ∪-shaped pattern in the intraday quote volume [27]. Such patterns are vaguely identified in Figure 4.1.1, which presents the intraday volume flow for select technology stocks as observed on June 21st 2012.



(a) AAPL *intraday volumes*

(b) AMZN *intraday volumes*

(c) INTC *intraday volumes*

(d) MSFT *intraday volumes*

Figure 4.1.1: The intraday volume profile for select technology stocks on June 21st, 2012. Orders are aggregated at every second and the heatmap is binned over 5 minute intervals.

The intraday volume flows play a critical role in any VWAP trading strategy, and understanding why follows directly from the ex-post calculation of VWAP. Formally, the Volume-Weighted Average Price for a security over the time horizon $[0, \tau]$ follows the straight-forward calculation

$$\text{VWAP}_{[0,\tau]} = \frac{\sum_{t\in[0,\tau]} p_t v_t}{\sum_{t\in[0,\tau]} v_t}$$

where p_t, v_t are the prices and respective volumes executed at each time t. It is appropriate to attribute a Market VWAP – in which case the prices and volumes used for the computation aggregate from what is executed in the exchange – just as it is appropriate to attribute a strategy-specific VWAP – then, prices and volumes come from the given strategy. The trader's *execution cost* is typically how VWAP is referred to on a strategy-basis.

When execution costs meet or best the Market VWAP, then the trading strategy is usually profitable. To understand why, consider a hypothetical example where a pension fund is looking to sell shares of a security which they hold a large inventory of. The pension fund itself may not be well-positioned as a dealer to execute this large trade in the market. Rather than directly incur punitive transaction costs on the order, the pension fund can appeal to the services of a brokerage firm to efficiently execute the order on their behalf. The brokerage firm would commonly acquire the position on a liability basis – that is, acquiring the position outright – and charge fees on a per-share basis. When the fee structure is directly tied to VWAP, then the brokerage firm has a clear profit objective: *execute the position at an average price that exceeds VWAP by as large of a margin as possible.* In this example, the pension fund is satisfied that their position is alleviated, and the brokerage firm can make a profit with an effective VWAP trading strategy.

A standard approach to the VWAP strategy comes from estimating an expected volume profile over the execution horizon and proportionately executing orders to match. A primer for VWAP strategies is given in [57].

VWAP execution strategies are studied under a transient price impact model, similar to that from chapter 3, in [8]. A more general stochastic model is considered in [18], where findings present a closed-form VWAP strategy that intrinsically tracks TWAP with appropriate offsets accounting for order-flow. Owing to the simple breakdown of the optimal VWAP strategy from [18], we will incorporate TWAP as a component in the reward signal when we train our A3C model to learn VWAP execution. Having said that, we now proceed to briefly introduce related applications of Machine Learning and Deep Reinforcement Learning to a financial setting.

4.1.2 Applications of Machine Learning to Order Execution

Machine Learning methods are now commonly applied to topics in Quantitative Finance. In terms of predicting price movements, regression techniques such as that from [99] form a baseline which can be extended to Feedforward [55], Recurrent [25] and Convolutional [98] Neural Network structures.

In the case of [98], with limit order book features it is shown that convolutional networks can apply a feature-mapping that recovers a measure for the order book imbalance. Order book imbalance calculates the normalized quote-volumes at a particular level in the order book. By specifying a limit order book of depth N by the set of quad-tuples $\{(p_b^{(l)}, v_b^{(l)}, p_a^{(l)}, v_a^{(l)})\}_{l=0}^{N}$, the feature-mapping outputs a micro-price $\tilde{p}^{(l)}$ for the l^{th} level in the order book based on

$$\tilde{p}^{(l)} = \frac{v_b^{(l)}}{v_b^{(l)} + v_a^{(l)}} p_a^{(l)} + \frac{v_a^{(l)}}{v_b^{(l)} + v_a^{(l)}} p_b^{(l)}$$

Order book imbalance is treated as a technical indicator for the direction of price-movements and helps inform various kinds of trading strategies [54] [74]. We appeal to the convenient economic interpretation and utilize convolutional layers in our A3C model.

Significant work in the space of feature-normalization has led researchers in [64] to produce a limit order book dataset, based on Nordic stocks, that is specifically designed for Machine Learning approaches. Shown further in [66] is a method for an adaptive normalization technique which is shown to sufficiently account for non-stationary data and produce stable predictions. We take these results as evidence in favour of a feature-normalization technique and adopt a simple version.

Shifting focus now to Reinforcement Learning applications, the pioneering work from [62] demonstrated that a variant of Q-Learning can train an agent to effectively set prices, and impart up to a 12.9% reduction in total transaction costs over comparable "submit-and-leave" orders. Of further significance is the work from [39], where the famous *Almgren and Chriss model* was cast into one compatible with Q-Learning. Here, the authors attempt *Implementation Shortfall* – a strategy which minimizes *slippage* – and credit profits that are on average 10% larger than the baseline Almgren and Chriss framework. Extending consideration to risk-averse Reinforcement Learning, material from [81] derives an action-value update rule that incorporates a concave utility function. The experiment performed in [81] demonstrated that an agent trained under the risk-averse Reinforcement Learning approach was able to maintain low transaction costs during the exceptional "Flash-Crash" market event observed on May 6th 2010, and noted an otherwise muted response when compared to a risk-neutral agent.

Further academic extensions of Deep Reinforcement Learning applied to Order Execution remain relatively few in number. Noteworthy is the contributions of [63], for first adapting a *Double-Deep Q-Network* that trains for Optimal Order Execution. Given the results indicating overall outperformance, we are encouraged to adapt an A3C model to target VWAP execution. To the best of our knowledge, there has been no academic contributions of A3C approaches to this objective. We then will carefully detail our approaches in the upcoming section.

4.2 Methodology

In the content to follow, we introduce three environments for a limit order book that facilitate the development of an A3C application to Optimal Order Execution. Brief comments are provided which outline the setup for each environment and explain the comparative advantages and disadvantages of each. Open-source code is provided and we encourage the continued development and modification of this code.

We settle on a real-time simulator for the development of our A3C model. We then proceed to introduce the baseline Neural Network structure, as highlighted by two convolutional layers and an LSTM cell. The training scheme is summarized by addressing the relevant input features, formulating the action space and designing the reward signal that completes our approach.

We conclude the methodology section by addressing a model for an accompanying ASD learner that can enhance future training approaches. Due to time constraints with this project, we cannot implement this approach with our work here.

Now, beginning on a remark detailing our computing system.

4.2.1 Computing System Configurations

All models in this chapter are trained using a shared Windows 2012 Server with an Intel® Xeon® Processor E5-2697 v4. There are 36 threads available to run the A3C model, although no more than 16 are used. No GPU's are used for training since the environment that we train our model on proves unconducive. Precisely why relates to a training scheme that is performed in real time, with latency delays that negate any performance benefit from using a GPU.

4.2.2 Revisiting the Block Shaped Limit Order Book

In section 3.2, we developed a model for the block shaped limit order book. Now, we work this model into one that is compatible for a Reinforcement Learning application. Consider the transient price impact model for an asset's bid- and ask-price

$$dA_t = dP_t + \lambda d\gamma_t^+ - \alpha(A_t - P_t)dt$$
$$dB_t = dP_t - \lambda d\gamma_t^- - \alpha(B_t - P_t)dt$$

Where $\gamma_t^\pm$ provides the quantity of shares purchased $(+)$ or sold $(-)$; functions which are non-decreasing and right-continuous with left limits. Further, $\lambda > 0$ provides the market depth factor which imparts the cost of a trade and $\alpha > 0$ gives the rate of recovery for the market. $(P)_{t\geq 0}$ is the fundamental price process, which may be taken as a Brownian Motion or reserved as a price observation. Both equations for the bid- and ask-prices can be solved, to give

$$B_{t+\delta t} = P_{t+\delta t} + (B_t - P_t)e^{-\alpha\delta t} - \lambda \int_{[t,t+\delta t]} e^{-\alpha(t+\delta t-s)} d\gamma_s^- \tag{4.2.1}$$

$$A_{t+\delta t} = P_{t+\delta t} + (A_t - P_t)e^{-\alpha\delta t} + \lambda \int_{[t,t+\delta t]} e^{-\alpha(t+\delta t-s)} d\gamma_s^+ \tag{4.2.2}$$

For each instance of time t within the trading horizon and for any arbitrary time-step δt. Then, Equation 4.2.1 (resp. Equation 4.2.2) can be taken as the resulting price dynamics for an agent looking to liquidate (resp. acquire) an inventory of shares. Briefly describing a practical setting, trading would take place at discrete points over a given time horizon, and the agent must decide how much of her inventory should be traded at each point in time. The agent would accordingly post a market order, recover the profits from the trade, and observe an impacted price state.

Under a practical approach, the integration in Equation 4.2.1 and Equation 4.2.2 becomes a summation, and transaction costs are levied in direct proportion to the quantity traded. Since the block shaped limit order book assumes uniform density beyond the best available bid- or ask-price, there is no support for an agent posting limit orders.

The main advantage to modelling with the block shaped limit order book comes from simplicity – the dynamics are convenient to specify and the model offers flexibility in terms of how $(P)_{t\geq 0}$ is defined. Unfortunately, the simplicity of the model also means that it can be a far departure from a real limit order book, lessening the relevance of application and posing problems for Zero-shot Learning, as discussed in background section 2.1.4. The approach may be appropriate for sufficiently liquid securities.

Since we have previously studied Order Execution on the block shaped limit order book in chapter 3, we do not use this environment for an application of Deep Reinforcement Learning. We do however provide source code[1] for an adaptation that extends *OpenAI*'s "Gym" framework, which greatly simplifies the training scheme.

4.2.3 Reconstructing the Historical Limit Order Book

When training on a modelled limit order book is deemed inappropriate, attention shifts towards using historical data. Several exchanges sell historical data from their limit order books, including Nasdaq's *TotalView* [61] and the New York Stock Exchange's *Open Book Ultra* [26]. The data coming directly from a stock exchange provides a rich foundation

[1] OpenAI "Gym" Environment for the block shaped limit order book: https://github.com/mbreiter/drloe_block

to study Market Microstructure, but drawing an application is challenged because the pricing is expensive for small-scale use and even still, the limit order book needs to be *reconstructed*. Addressing both of these challenges is the LOBSTER service[2].

Limit Order Book System - The Efficient Reconstructor, or *LOBSTER* for short, is an academic resource that provides high quality historical limit order book data from Nasdaq's TotalView. LOBSTER is commonly used as a data provider for empirical studies on the limit order book; [38] for example. While there is a subscription fee to access LOBSTER, five free data samples are provided for select technology stocks – *MSFT*, *INTC*, *AAPL*, *GOOG* and *AMZN* – that motivate our adaptation of a compatible environment for Reinforcement Learning. The free data, dated for June 21st 2012, provides bid- and ask-quotes up to ten levels deep, and is given at a millisecond resolution throughout the entire trading day. Figure 4.2.1 shows how real orders are distributed in the limit order book for *MSFT*, at approximately an hour before market-close. Further, Table 4.2.1 provides an intraday average for the *Bid/Ask Spread* and the *Volume Imbalance*.

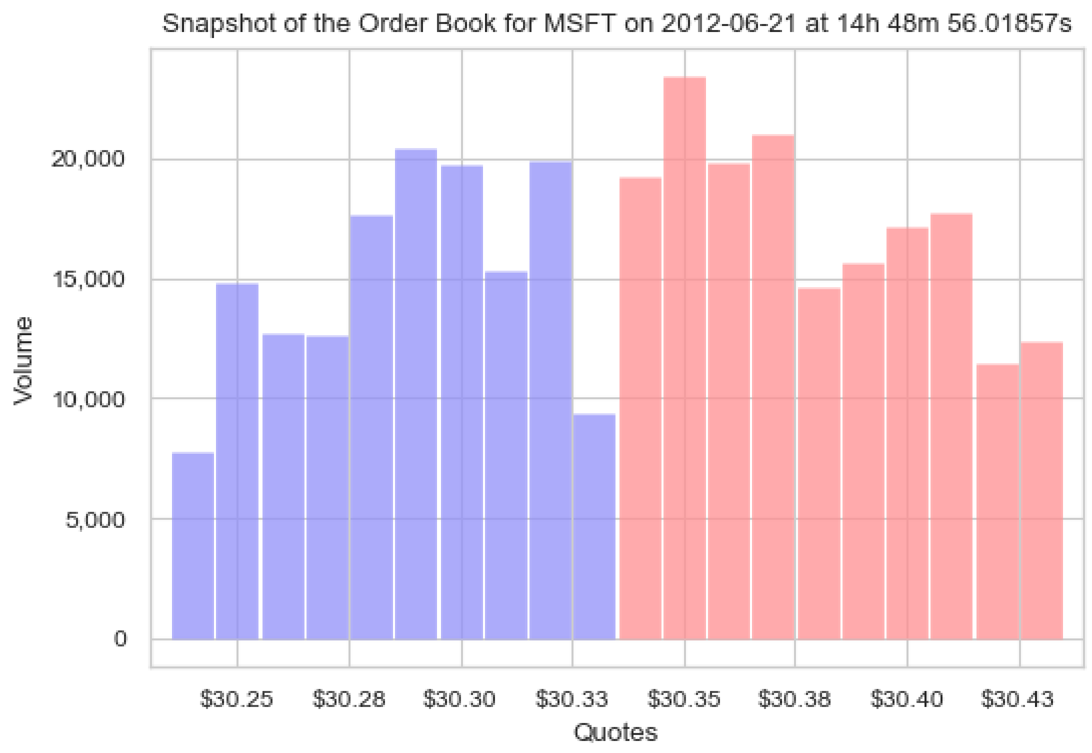


Figure 4.2.1: The Limit Order Book for MSFT on June 21st, 2012 at 14h 48m 56s.

Our methodology for constructing an environment is described as follows. We reserve four stocks for training – *INTC*, *AAPL*, *GOOG*, *AMZN* – and designate *MSFT* for testing. We consider only short execution horizons, lasting one minute, to maximize the number of unique, *but not uncorrelated*, trajectories that we can generate from a single

[2]LOBSTER Academic Data: `https://lobsterdata.com/info/WhatIsLOBSTER.php`

Table 4.2.1: Order Book characteristics for select technology stocks on June 21st, 2012.

Ticker	Average Bid/Ask Spread ($)	Average Imbalance
AAPL	0.1535	-0.0697
AMZN	0.1309	-0.103
INTC	0.0123	-0.0601
GOOG	0.2962	0.0005
MSFT	0.0123	0.0288

days worth of data. We then randomly query a starting time throughout the day and initialize the agent's state on the limit order book as rendered. The interpretation of this setup is that a trader may receive an order request from a client at any point during the day. The agent then begins trading, progressing through each time indexed state of the limit order book. We utilize a simple order matching engine that immediately fills active orders and queues any remaining passive component at an appropriate level in the order book. The state of the order book is updated at each training episode, but we persist any of the agent's outstanding orders and execute these on a time-priority basis. Further, we approximate an agent's market impact by preprocessing each time indexed state of the order book to filter out any quotes that would have already been filled by the agent's executed trades. This is an approximation to market impact because, by using historical data, we fundamentally rest on the assumption that the agent's trading activity does not influence price evolution. In a real market, if a reckless trader is moving prices significantly, then new orders would be posted in response. While our approach cannot account for this real-life dynamic, we are able to ensure that the agent is exposed to the effects of her own trading. Trading as outlined continues until the time horizon is spent, in which case we reset by randomly picking another starting time and beginning anew.

In our application, we are severely limited by only having data for a single day. We cannot reasonably justify that the stocks are uncorrelated – *they certainly would be correlated* – and we cannot generalize further beyond short execution windows, where price evolution would play a diminished role in the execution strategy. For these reasons, we do not progress our study with the LOBSTER-based environment but make available the source code for those who might have a subscription to LOBSTER[3].

Finally, we present the environment which we progress our application on.

4.2.4 Trading in a Simulated Stock Environment

The Rotman School of Management at the University of Toronto maintains the *Rotman Interactive Trader*[4] (RIT) application, which is a comprehensive simulation of an order-

[3]OpenAI "Gym" Environment for a LOBSTER-based limit order book: https://github.com/mbreiter/drloe_lobster

[4]The website maintaining resources for the RIT application: http://rit.rotman.utoronto.ca/

driven market. RIT enables interactive trading in real-time and maintains a centralized limit order book that all participants on the server can view and post orders to. Furthermore, the underlying price diffusion for a security in RIT is a Geometric Brownian Motion, from which the price path is supported, in the absence of market participants, by computerized agents that scatter the order book with quotes.

As a rich educational resource, RIT also proves compatible to train a Reinforcement Learning model for VWAP execution. Trading on RIT is carried-out in “heats”, each lasting approximately five minutes. The parameters stipulating liquidity conditions are configurable, so we choose to build our environment for a security that has ample liquidity. Furthermore, we allow both a modest drift and volatility factor to characterize the price path of the security that we trade in RIT.

Our methodology for constructing an environment on RIT is described as follows. We instantiate a connection to RIT after opening the application and configuring the ports which facilitate all communication. We then allow an agent to post limit orders by sending API requests to the RIT server, stipulating the details of the order. The response to the agent is returned as the state of the market, with additional details such as inventory and price levels in the order book. Trading in this fashion continues until a predefined terminal time is reached, after which the environment is paused until the RIT server restarts and strikes again the starting point. Since RIT maintains the centralized limit order book, our adaptation of the Reinforcement Learning environment is greatly simplified and only requires caution when handling the API requests.

What is immediately worth noting about RIT is that the server instance is *shared*. This poses an interesting twist to our A3C model which, as described in background section 2.1.7, trains with distributed processes. The result of this means that all agents in the distributed network are exposed to the actions of one another, whereas typically, these environments are kept separated. From this perspective, the shared environment mimics a setting with a Nash equilibrium. All that to say, we are attuned to this effect.

We chose to train our A3C model on the environment adapted to RIT, justifying that the simulation engine offers a sufficiently dynamic limit order book that extends beyond the transient price impact model. The source code for the Open AI “Gym” extension is made available at a public repository[5].

Having described the training environments, allow us now to introduce our model structure.

4.2.5 Training Scheme for the Baseline Model

Figure 4.2.2 presents the structure of the Neural Network which is used to support VWAP execution. In the content to follow, we justify the design considerations for the network, and discuss the training scheme.

[5]OpenAI “Gym” Environment for RIT: https://github.com/mbreiter/drloe_rit

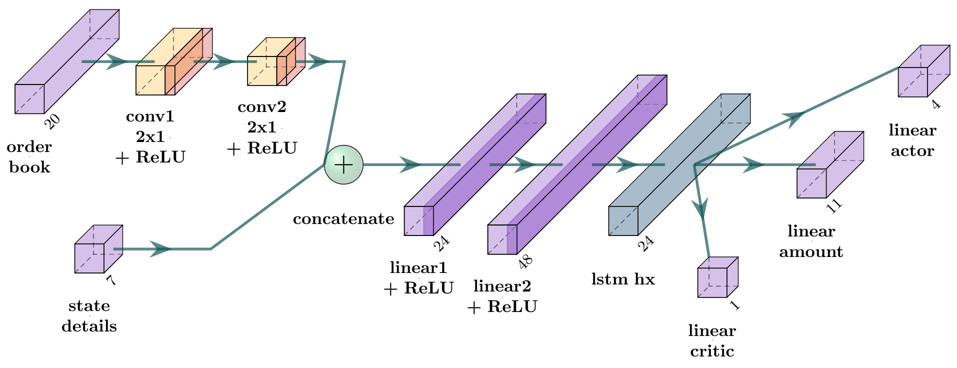


Figure 4.2.2: The Neural Network designed for VWAP execution.

4.2.5.1 Input Features

As per Figure 4.2.2, the input space consists of 27 features taken in the form of two components: the first capturing the limit order book data and the second providing relevant state details. These features are summarized below:

$$\begin{aligned}
\textbf{Limit Order Book Features}: \quad & \left\{\left(\frac{p_b^{(l)}}{N_p}, \frac{v_b^{(l)}}{N_v}, \frac{p_a^{(l)}}{N_p}, \frac{v_a^{(l)}}{N_v}\right)\right\}_{l=0}^{4} \in \mathbb{R}^{20} \\
\textbf{State Detail Features}: \quad & \left(\frac{\text{COST}_{[t_{\text{start}},t]}}{N_p}, \frac{\text{VWAP}_{[t_{\text{start}},t]}}{N_p}, \right. \\
& \qquad \left. \frac{t - t_{\text{start}}}{t_{\text{end}} - t_{\text{start}}}, \frac{I}{N_\rho}, \frac{\rho_t}{N_\rho}, \frac{\tilde{\rho}_t}{N_\rho}, \mathbb{1}_{\{t=t_{\text{end}}\}}\right) \in \mathbb{R}^7
\end{aligned}$$

Where $p_b^{(l)}$, $p_a^{(l)}$ are the bid- and ask-prices for the l^{th} level in the limit order book; $v_b^{(l)}$, $v_a^{(l)}$ are the bid- and ask-volumes for the l^{th} level in the limit order book; I is the inventory that the agent must liquidate (resp. acquire); ρ_t is the position at time t which has been executed; $\tilde{\rho}_t$ is the magnitude of any pending orders at time t; and N_p, N_v, N_ρ are normalization factors for the prices, volumes and positions respectively.

We carefully consider which state details should be fed into the network. It is important that the model is exposed to both the agent's COST and VWAP, since we seek a profit maximizing strategy based on these endogenous variables. Further, the agent should also preserve some sense towards her position executed and amount left outstanding, so as to inform decisions pertaining to the tradeoffs between guaranteed price execution and favourable price-setting. The time-based features are included so the agent is aware of the constraint requiring execution to be completed by the end of the time horizon.

Heeding the findings from [17], which attributes 78% of price discovery to the best bid- and ask-prices, we consider only the first 5 levels in the limit order book. By narrowing our focus on the limit order book, we are able to reduce the dimension of the input space without sacrificing any significant information loss.

The normalization techniques mentioned in section 4.1 are simplified to the point where only constant factors are considered. For the present application, we take $N_p = 100$ and $N_v = N_\rho = 10I$. It is worth noting that the price of the security traded in the environment always begins at \$25.00 and never exceeds a price of \$30.00, meaning all input features are within the unit range $[0, 1]$.

Figure 4.2.2 depicts the order book features passing through two sequential convolutional layers. Each convolutional layer is one-dimensional with a (2×1) kernel and no stride nor padding is applied. When passed sequentially in such a manner, the output result is a tensor with dimension reduced by a factor of four. Recalling that convolutional layers apply a feature-mapping that is akin to an imbalance measure, as discussed in section 4.1, the output from the convolutional layers can be treated as a technical indicator for each price level in the order book. The output from the convolutional layers is concatenated with the state detail features and the combined tensor is then subjected to a familiar LSTM network.

Having discussed the structure of our network right up to the output layer, we now address the action space that we permit for our agent.

4.2.5.2 Action Space Considerations

The model presented in Figure 4.2.2 trades by specifying an action tuple $(a_t^{(T)}, a_t^{(Q)})$, where $a_t^{(T)}$ gives the trade type and $a_t^{(Q)}$ gives the quantity to be traded. In a general sense, a trader can submit a wide variety of order types, and she can indeed choose to trade in any allotment size she wishes, provided that the volume is available in her unspent inventory. Recognizing that these general cases greatly expand the size of the action space, we take necessary steps to simplify the model. But first, we acknowledge the class of trading strategies which we restrict our focus to.

Remark 4.2.1 *We only consider VWAP strategies which are unidirectional; an agent cannot deviate from her objective and purchase (resp. liquidate) when she is attempting share liquidation (resp. acquisition). This condition becomes relevant for large firms who are bound by strict regulations governing market manipulation. Conceivably, an unbounded Reinforcement Learning agent might very well learn to trade a policy that involves manipulating prices.*

There are four types of trades that our agent can decide on, given in the following set $\mathcal{A}_T = \{$`HOLD, BBO, MID-QUOTE, ACTIVE`$\}$. These actions have the following interpretations:

`HOLD`: calls for no trade to be executed;

BBO: connotes a limit order posted at the best bid- or ask-price;

MID-QUOTE: outlines a limit order posted at the mid-quote;

ACTIVE: gives an active limit order, but unlike a market order, this type of limit order will not walk-through the order book;

In addition to the trade types given by $\mathcal{A}_T$, which are always available for the agent to act on, a MARKET-ORDER is executed at the end of the trading period if inventory is left unspent. This clearing action effectively imposes a punitive charge, brought about by a large market impact, if the agent does not complete her execution within the horizon.

The agent is not permitted to CANCEL her outstanding orders. A possible extension of this framework might elect to include this action type, but we omit consideration on account of the added complexity. What's more, with the reward signal that we intend to craft – that which will incentivize the tracking of TWAP – we anticipate that few orders will be left outstanding.

Further, the agent can choose to trade quantities given by the percentages in following set $\mathcal{A}_Q = \{i/200\}_{i=0}^{10}$. These percentages, ranging from 0% to 5%, are interpreted as quantities in view of the agent's initial inventory position. For example, if the agent must liquidate 10,000 shares and it is decided that $a_0^{(Q)} = 0.035$, then 350 shares would be traded. We feel justified in considering a reduced action-quantity space on account of the TWAP tracking incentive in the reward signal; seldom should it be appropriate for an agent to trade in large block sizes throughout the horizon. Furthermore, a reduced action space is favourable given the computation limitations.

Together, the combined action space $\mathcal{A} = \mathcal{A}_T \times \mathcal{A}_Q$ permits 44 unique action tuples. We now mention how the agent's actions earn a reward.

4.2.5.3 Reward Signal Engineering

We draw inspiration from [18] to craft a reward signal that incorporates a TWAP tracking component. The reward signal is given below, and is defined for both a liquidation $(-)$ and an acquisition $(+)$ objective

$$r^{\pm}(t) = \underbrace{\mathbb{1}_{\left\{\left|\frac{\rho_t}{I} - \frac{t - t_{\text{start}}}{t_{\text{end}} - t_{\text{start}}}\right| \leq \epsilon\right\}} - 0.5}_{\text{incentive to track TWAP}} + \underbrace{\mathbb{1}_{\{\rho_{\text{t}-1} \neq \rho_{\text{t}}\}} \mathbb{1}_{\left\{\mp\left(\text{COST}_{[t_{\text{start}}, t]} - \text{VWAP}_{[t_{\text{start}}, t]}\right) \geq 0\right\}}}_{\text{incentive to outperform VWAP}} \tag{4.2.3}$$

Where $\epsilon \in [0, 1]$ gives a tolerance parameter for deviating from TWAP, which we take as $\epsilon = 0.05$. As per Equation 4.2.3, the agent is incentivized to keep her position close to the time-progression status throughout the horizon.

The VWAP incentive is realized whenever an order is executed and imparts a unit-reward if the agent's COST at time t meets or exceeds the Market-VWAP at the time of execution. Having introduced the reward signal, we will next remark on the training scheme.

4.2.5.4 Distributed Training Approach

The A3C model is run with 16 training processes. Each training agent is assigned, uniformly at random, either a liquidation or acquisition directive. The following parameters are also assigned randomly based on the discrete uniform distribution $\mathcal{U}(a,b)$: $I = 10,000\Big(10+\mathcal{U}(-4,4)\Big)$, $t_{\text{start}} = 5+\mathcal{U}(0,75)$ and $t_{\text{end}} = 290-\mathcal{U}(0,100)$. Algorithm 3 outlines the adapted A3C algorithm that facilitates VWAP execution.

Algorithm 3 Asynchronous Advantage Actor Critic for VWAP execution

```
Inherit shared global parameters $\theta_T$, $\theta_Q$, $\theta_v$ and common global counter $\tau = 0$
Initialize process-specific parameters $\theta'_T$, $\theta'_Q$, $\theta'_v$
Initialize process step counter $t \leftarrow 1$
repeat
    Reset gradients $d\theta_T \leftarrow 0$, $d\theta_Q \leftarrow 0$, $d\theta_v \leftarrow 0$
    Synchronize process-specific parameters $\theta'_T \leftarrow \theta_T$, $\theta'_Q \leftarrow \theta_Q$, $\theta'_v \leftarrow \theta_v$
    $t_{\text{start}} \leftarrow t$
    repeat
        Choose trade type $a_t^{(T)}$ and trade quantity $a_t^{(Q)}$ from $s_t$ using $\pi$
        Compute entropies $e_t^{(T)}$, $e_t^{(Q)}$
        Execute trade $\Big(a_t^{(T)}, a_t^{(Q)}\Big)$, observe new state $s_{t+1}$
        construct reward $r_t$ according to Equation 4.2.3
        $t \leftarrow t+1$
        $\tau \leftarrow \tau+1$
    until $s_t$ is terminal or $t - t_{\text{start}}$ reaches the maximum step count
    $R = \begin{cases} 0 & s_t \text{ terminal} \\ V(s_t, \theta'_v) & \text{otherwise} \end{cases}$
    Initialize Generalized Advantage Estimator GAE $\leftarrow 0$
    $i \leftarrow t-1$
    repeat
        $R \leftarrow r_i + \gamma R$
        $\text{GAE} \leftarrow \tau\gamma\text{GAE} + r_i + \gamma V(s_{i+1}, \theta'_v) - V(s_i, \theta'_v)$
        $d\theta_T \leftarrow d\theta_T + \nabla_{\theta'_T}\Big[\log\pi(a_i|s_i;\theta'_T,\theta'_Q)\text{GAE} - 0.01e_i^{(T)}\Big]$
        $d\theta_Q \leftarrow d\theta_Q + \nabla_{\theta'_Q}\Big[\log\pi(a_i|s_i;\theta'_T,\theta'_Q)\text{GAE} - 0.01e_i^{(Q)}\Big]$
        $d\theta_v \leftarrow d\theta_v + \frac{\partial}{\partial\theta'_v}\Big(R - V(s_i, \theta'_v)\Big)^2$
        $i \leftarrow i-1$
    until $i < t_{\text{start}}$
    Asynchronous updates of shared global parameters
until $\tau >$ maximum time for the episode
```

4.2.6 Extensions to an ASD Learner

In Figure 4.2.3, a Neural Network model is presented that is conducive to an ASD learning framework. In section 4.4, we discuss how such a framework can build on the results we now present for VWAP execution.

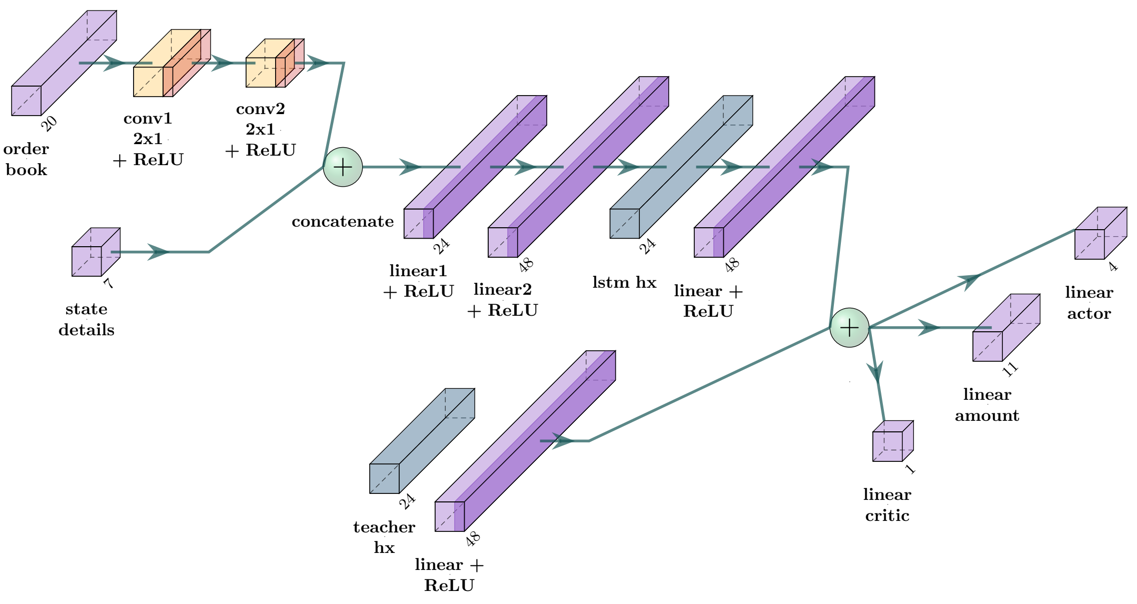


Figure 4.2.3: The Advice & State Driven scheme to train a Student for VWAP Execution.

4.2.7 Performance Evaluation

The testing scenario is conducted for an agent seeking liquidation, with an inventory of $I = 100,000$, a start-time of $t_{\text{start}} = 5$, and an end-time of $t_{\text{end}} = 290$.

Our objective is to train the A3C model to execute at prices that meet or exceed VWAP. In evaluating the performance of the model, we take the criterion that profits are calculated as the excess between the cost $\text{COST}_{[t_{\text{start}},t_{\text{end}}]}$, and VWAP with a one-cent discount, $\text{VWAP}_{[t_{\text{start}},t_{\text{end}}]} - 0.01$. Therefore, a trading strategy that outperforms VWAP would post profits exceeding ten basis points.

4.3 Main Results

Training consisted of approximately 1700 episodes over the course of one week. Figure 4.3.1 shows the learning curve, with a view of the agent's episodic action distributions and quantity allocations. This figure becomes the subject for immediate analysis and discussion.

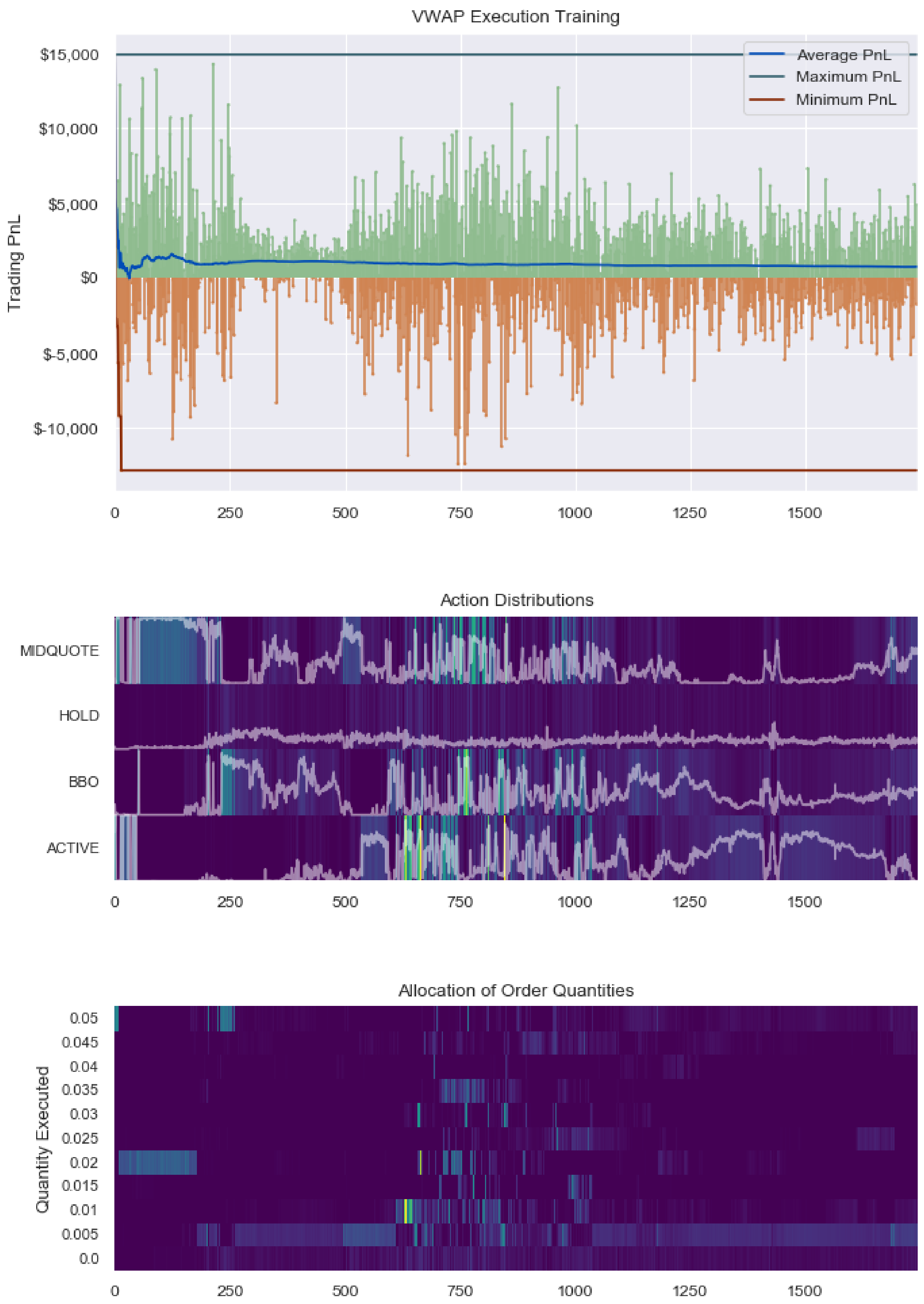


Figure 4.3.1: The training curve, action distributions and quantity allocations for the baseline VWAP execution model.

Throughout the training period, the agent posted a modest profit, but only after including the one-cent discount from VWAP. When it came to tracking VWAP throughout the training session, the agent underperformed by three basis-points on average.

Figure 4.3.1 provides insight into the learning progression of our agent. Within the first 200 training episodes, the agent primarily posted limit orders at the mid-quote, and traded almost exclusively in 2% allotment sizes. The result led to the most volatile profit-levels realized throughout the training, as the agent traded through her inventory early-on and was detrimentally unable to capture price appreciations towards the end of the trading horizon. Profits realized during this early period of training are not attributed to an effective execution.

For a short period, from about the 250th episode through to the 500th, the agent began trading in smaller quantities and at more favourable prices. During this period, the agent outperformed VWAP consistently. However, as the agent continued to explore her action space, active orders became more frequent and larger swings in profits followed.

The agent appears to have learned to track TWAP, as evidenced by the large concentration of small order quantities. This would suggest that the TWAP tracking factor dominates the VWAP outperformance incentive in Equation 4.2.3. In section 4.4, we recommend adjusting the reward signal to better incentivize the agent to outperform VWAP.

To better understand the actions of our agent in view of underlying market conditions, we devise a testing experiment to compare the actions taken by the agent against the price path of the security. Figure 4.3.2 presents the results from a testing period consisting of 140 episodes. The agent observes a small profit – again with the one-cent discount to VWAP – and selects her actions very consistently throughout the test. This includes a clear bias towards trading in 0.5% allotment sizes.

To arrive at a better understanding as to how our agent is trading in the environment, we study two episodes in Figure 4.3.3. The 139th episode was the most profitable for the agent, and it becomes clear that this profit was earned by densely liquidating inventory within the first half of the trading window. In contrast, the 87th episode brought trading losses which amounted because the agent did not capture the upward price trend towards the end of the horizon.

Results from Figure 4.3.3 suggest that the agent was able infer trends with the price path, since even in the unprofitable episode, the agent abstained from trading when the price was low at the beginning of the horizon. To this effect, the agent appears to struggle with the timing of her trades. We hypothesize that more training would be required for the agent to learn how to distribute her orders throughout the trading horizon.

We use these findings to inform recommendations for next-steps and future work.

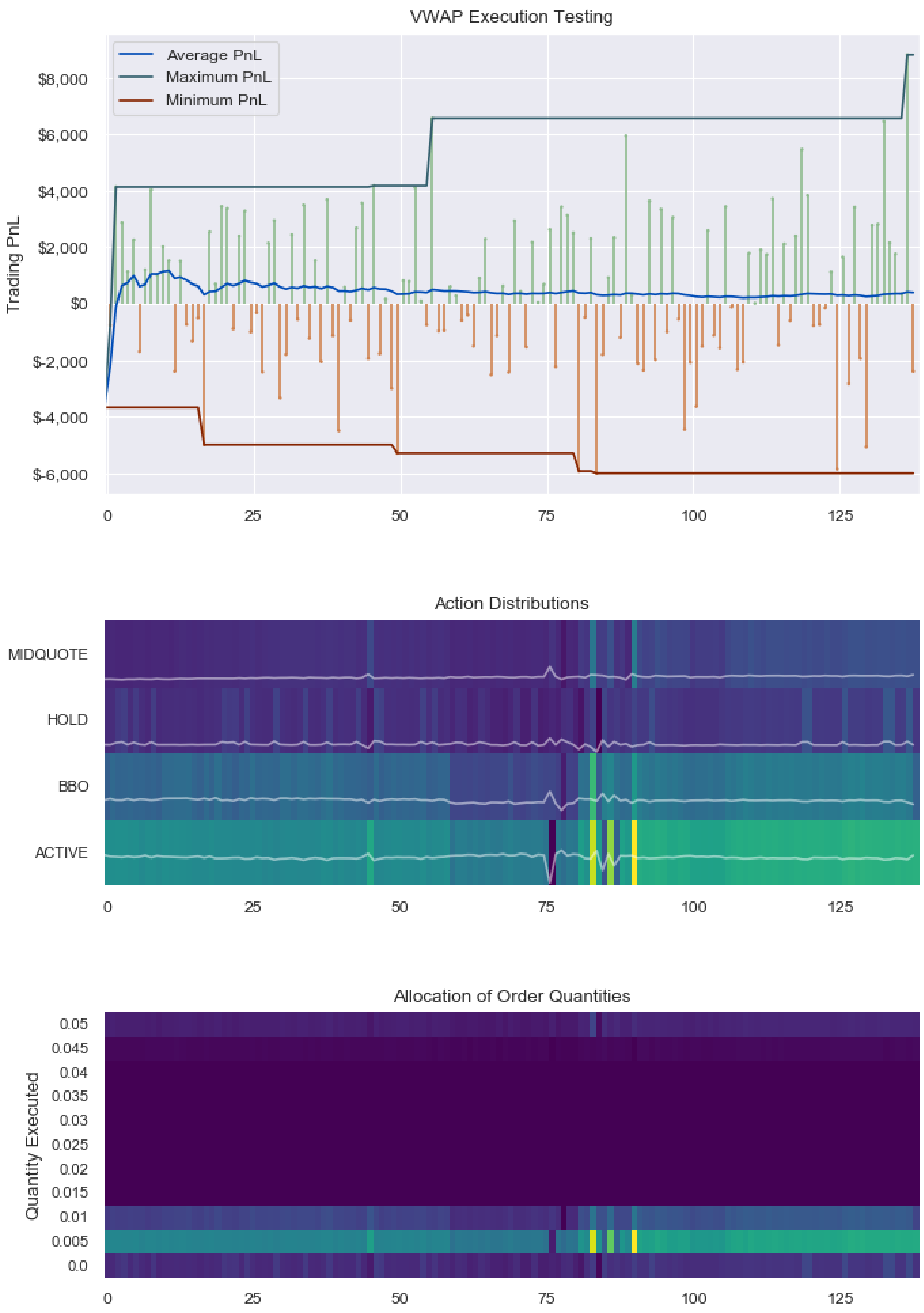
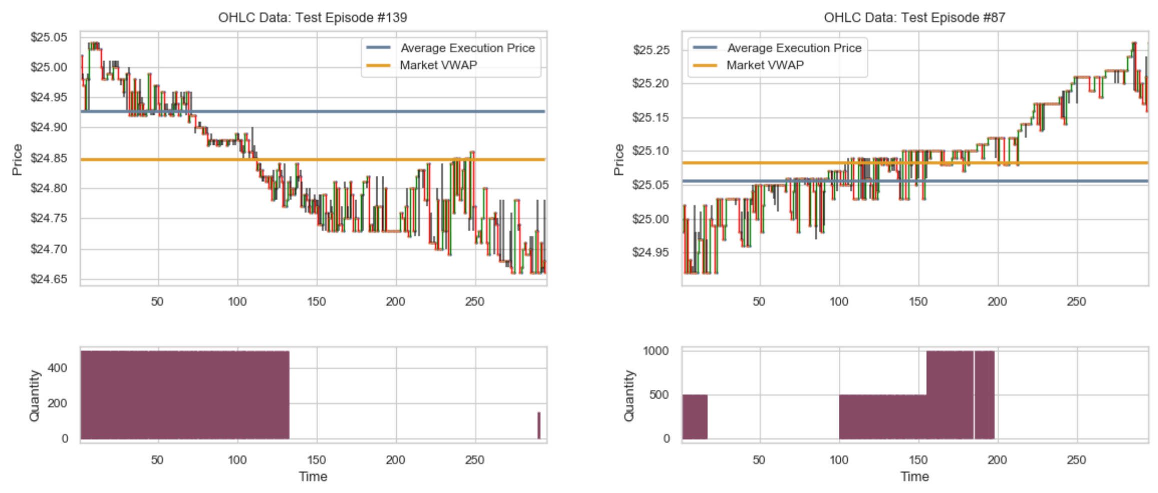


Figure 4.3.2: The testing results for the baseline VWAP execution model.



(a) A profitable scenario for the agent. *(b) A unprofitable scenario for the agent.*

Figure 4.3.3: Performance of the VWAP execution agent in view of market dynamics.

4.4 Future Work

The results do not overwhelmingly indicate that the agent was successful in beating VWAP consistently. There was evidence that the agent responded to the TWAP incentive, but it remains that the agent did not effectively time her trades throughout the horizon. Based on the performance, we acknowledge three next-step approaches:

▶ **Next-step 1:** We hypothesize that the performance would improve if the agent is allowed to collect more experience. Given that the action space is large, consisting of 44 unique possible actions, it is conceivable that approximately 1700 episodes is insufficient to inform an agent how to set price-levels, determine quantity allotments and distribute orders appropriately throughout the trading horizon.

▶ **Next-step 2:** We suggest modifying the reward signal, Equation 4.2.3, to incorporate a stronger incentive for the agent to outperform VWAP. A proposed reward signal is given below wherein the agent would directly receive a reward based on the magnitude of outperformance with respect to VWAP.

$$r^{\pm}(t) = \underbrace{\mathbb{1}_{\left\{\left|\frac{\rho_t}{I} - \frac{t-t_{\text{start}}}{t_{\text{end}}-t_{\text{start}}}\right|\leq\epsilon\right\}} - 0.5}_{\text{incentive to track TWAP}} \mp \underbrace{100 \times \mathbb{1}_{\{\rho_{t-1}\neq\rho_t\}}\left(\text{COST}_{[t_{\text{start}},t]} - \text{VWAP}_{[t_{\text{start}},t]}\right)}_{\textbf{stronger}\text{ incentive to outperform VWAP}} \tag{4.4.1}$$

Whereas the previous formulation only offered a unit-reward for *outperformance*, Equation 4.4.1 provides negative reinforcement for any *underperformance*. The TWAP incentive persists, since the agent appeared to respond favourably to this signal.

▶ **Next-step 3:** With a modified reward signal comes an opportunity to adopt a Teacher/Student learning framework, as studied in chapter 2. Under an ASD learner, the agent trained under Equation 4.2.3 can act as a Teacher for a Student receiving reinforcement from Equation 4.4.1. This approach would capture the positive learning attributes observed in section 4.3, while focusing more on profit-maximization.

Indeed, we further motivate this continued work by presenting results from a small-scale training session for the ASD model. As shown in Figure 4.4.1 on the following page, the ASD learner was able to outperform VWAP by approximately 0.5 basis points. While we were limited in time for developing this model, the early profit indications give reason to believe that the modified reward signal from Equation 4.4.1 in tandem with the ASD learning framework can effectively be utilized for Order Execution.

While more extensive testing and analysis would call for a direct comparison between different trading strategies, in this work we have demonstrated the feasibility of applying a Deep Reinforcement Learning approach for Order Execution.

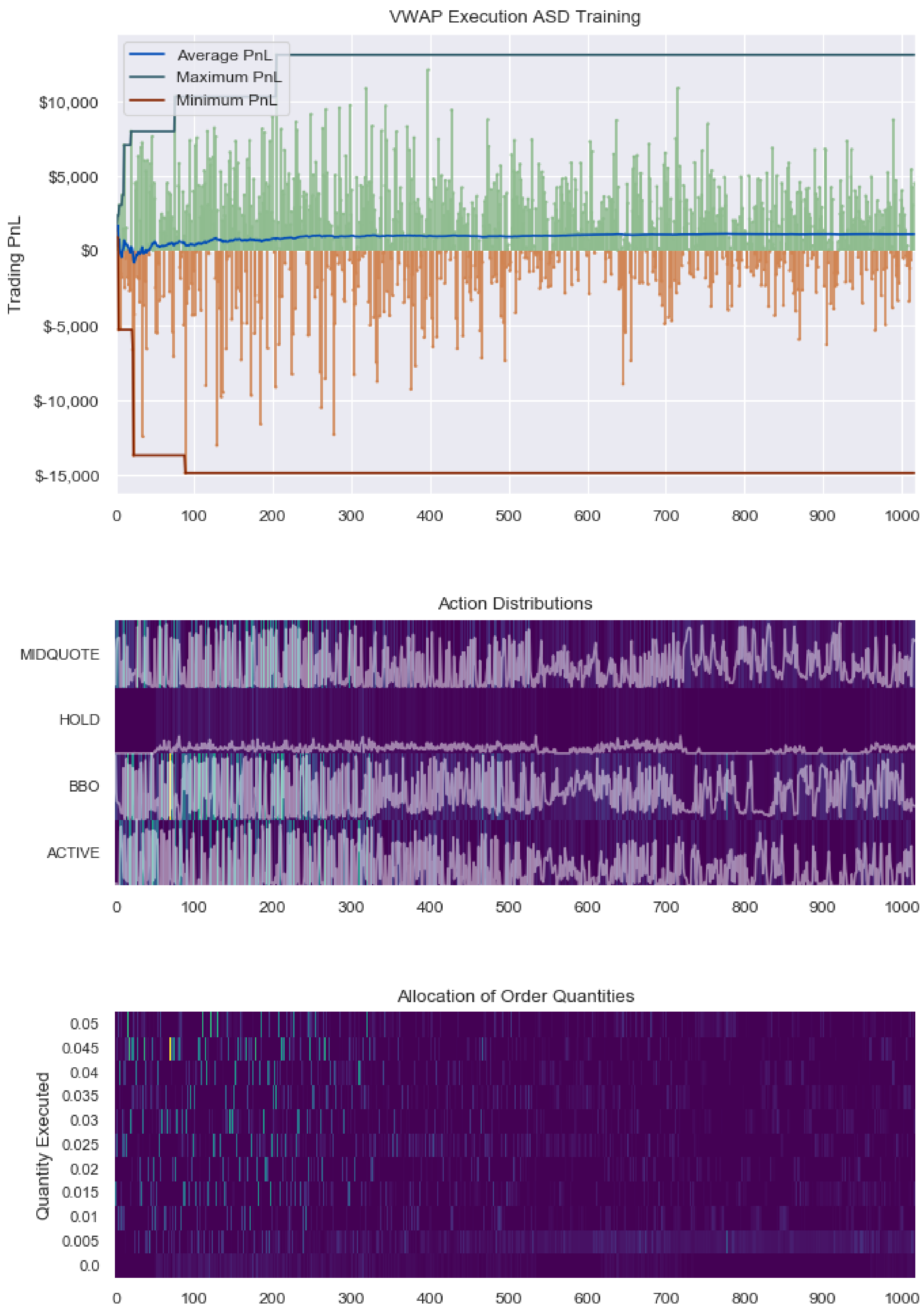


Figure 4.4.1: The training curve, action distributions and quantity allocations for the ASD VWAP execution model.

Chapter 5

Conclusion

This work has presented an application of Deep Reinforcement Learning for Volume-Weighted Average Price execution. In order to do so, two supporting objectives were introduced to address related concepts and methodologies that proved useful for the Order Execution theme. While each chapter concluded on a note providing extensions and/or recommendations for further work, we take the opportunity here to summarize our approaches and again motivate development.

The Teacher/Student learning framework from chapter 2 outlines an efficient approach to train Deep Reinforcement Learning models. The best performing Student model was the ASD learner with preinitialized weights from the Teacher. Two of the ASD models realized an average episode reward that was 3% (ref. Breakout) and 20% (ref. Beamrider) larger than their comparable preinitialized baseline models. While the results are encouraging, the ASD model for Space Invaders underperformed the baseline model by 6% on account of the Teacher being suboptimally trained. As a result, the engineering of the reward signal proves critical to the training scheme of Teacher model, and has a downstream effect on the learning ability of the Student.

The success of the Teacher/Student learning framework depends on the complexity of the environment, the relevance of the Teacher's advice for the Student's objective, and the level of mastery achieved by the Teacher. The Teacher/Student learning framework culminates to the following recommendation:

▶ **Recommendation 1:** The Teacher/Student framework is evidenced to train a reference Deep Reinforcement Learning model more efficiently. To leverage this framework, the parameters for the Student model should be preinitialized from the Teacher's network, and the structural component for the Student's network should be under the *Advice & State-Driven* decisioning approach. The resulting ASD learner would be equipped to learn enriched generalizations from the Teacher's advice, and efficiently reconcile this input with the environmental observation-stream.

The transient price impact model studied in chapter 3 provides an accessible entry point to the study of Optimal Order Execution. The accessibility comes from a solution method which avoids a complicated Hamilton-Jacobi-Bellman type equation, and instead uses Quadratic Programming and Variational Calculus. Continued work under this framework may be motivated by expanding the dimensionality of the model to cover multiple assets or by introducing a game-theoretic approach to study a Nash equilibrium for a market with competing agents.

The analysis of the discrete-time system offers meaningful insights into the influence of market depth, resilience, tightness and bias on the optimal trading policies. In light of the promising interpretations, the recommendation coming from this work is given:

▶ **Recommendation 2:** The block shaped limit order book is a convenient setting to define a transient price impact model on and outline an Optimal Liquidation objective to. The solution modelling in discrete-time is easily achieved as a Quadratic Program, and the dynamics allow for insightful continuous-time representation. The take-away from completing this work is a characterization of the richly diverse trading policies under varying agent-specific and market-specific parameters.

Where chapter 2 makes familiar the Deep Reinforcement Learning approaches, chapter 3 builds a relevant background for the Optimal Order Execution objective. From this perspective, the A3C model for VWAP execution from chapter 4 is built on the foundational work from the previous chapters.

The A3C model for VWAP execution is constructed from the ground-up, and involves the careful implementation of trading environments which are conducive for Reinforcement Learning applications. While the A3C model for VWAP execution remains simple, and the accompanying results prove modest, the extensive detailing of the setup offers ample room for improvements. Immediately, evidence shows that the ASD learning framework would be well suited for supplemental training efforts. Under this pretext, the final recommendation is summarized as:

▶ **Recommendation 3:** The A3C model is able to modestly track VWAP. The limitations of the modelling scheme introduce an opportunity to consider an enhanced reward signal, whereby an *Advice & State-Driven* learning model is a prime candidate for supplemental training.

Optimal Order Execution has challenged those who study, or practice, the subject to grasp the wonderfully complicated world of market microstructure. Just as Deep Reinforcement Learning will continue to develop in maturity, so too will the connection strengthen for applications to trading.

Bibliography

[1] Frédéric Abergel, Marouane Anane, Anirban Chakraborti, Aymen Jedidi, and Ioane Muni Toke. *Limit Order Books*. Physics of Society: Econophysics and Sociophysics. Cambridge University Press, 2016.

[2] Alex Graves et al. Speech Recognition with Deep Recurrent Neural Networks. International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2013.

[3] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems (NeurIPS), 2012.

[4] Anna A. Obizhaeva and Jiang Wang. Optimal trading strategy and supply/demand dynamics. *Journal of Financial Markets*, 16(1):1–32, 2013.

[5] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent Neural Networks Are Universal Approximators. Artificial Neural Networks – ICANN. Lecture Notes in Computer Science, vol 4131. Springer, Berlin, Heidelberg, 2006.

[6] Peter Auer, Harald Burgsteiner, and Wolfgang Maass. A learning rule for very simple universal approximators consisting of a single layer of perceptrons. *Neural Networks*, 21(5):786–795, 2008.

[7] Shmuel Baruch. Who benefits from an open limit-order book? *The Journal of Business*, 78(4):1267–1306, 2005.

[8] Alexander Barzykin and Fabrizio Lillo. Optimal vwap execution under transient price impact, 01 2019.

[9] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017.

[10] Dan Bernhardt and Eric Hughson. Splitting orders. *The Review of Financial Studies*, 10(1):69–101, 1997.

[11] Nataliya Bershova and Dmitry Rakhlin. The non-linear market impact of large trades: Evidence from buy-side order flow. *Quantitative Finance*, 13, 01 2013.

[12] Dimitris Bertsimas, Andrew W. Lo, and Paul Hummel. Optimal control of execution costs for portfolios. *Comput. Sci. Eng.*, 1(6):40–53, 1999.

[13] Bruno Bouchard, Masaaki Fukasawa, Martin Herdegen, and Johannes Muhle-Karbe. Equilibrium Returns with Transaction Costs. *Finance and Stochastics*, 22(3):569–601, 2018.

[14] Jean-Philippe Bouchaud, J. Doyne Farmer, and Fabrizio Lillo. How markets slowly digest changes in supply and demand. Papers 0809.0822, arXiv.org, September 2008.

[15] Brendan O'Donoghue et al. Combining Policy Gradient and Q-Learning. International Conference on Learning Representations, 2017.

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. cite arxiv:1606.01540.

[17] Charles Cao, Oliver Hansch, and Xiaoxin Wang. The information content of an open limit-order book. *Journal of Futures Markets*, 29(1):16–41, 2009.

[18] Álvaro Cartea and Sebastian Jaimungal. A closed-form execution strategy to target volume weighted average price. *SIAM J. Financial Math.*, 7(1):760–785, 2016.

[19] Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Conference on Empirical Methods in Natural Language Processing(EMNLP)*, 2014.

[20] Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and George E. Dahl. On empirical comparisons of optimizers for deep learning. 2019. cite arxiv:1910.05446.

[21] Rama Cont and Marvin Mueller. A stochastic pde model for limit order book dynamics, 04 2019.

[22] Csaba Szepesvari. Algorithms for Reinforcement Learning. In *Synthesis Lectures on Artificial Intelligence and Machine Learning.* Morgan Claypool Publishers, 2009.

[23] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams . Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[24] Ding-Xuan Zhou. Universality of Deep Convolutional Neural Networks. 2018.

[25] Matthew Dixon. High frequency market making with machine learning, 12 2016.

[26] The New York Stock Exchange. Openbook ultra client specification.

[27] P. N. Kolm F. J. Fabozzi, S. M. Focardi. *Quantitative equity investing - techniques and strategies.* 01 2010.

[28] Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning, 2018. cite arxiv:1811.12560.

[29] Masaaki Fukasawa. Conservative delta hedging under transaction costs. 03 2011.

[30] Masaaki Fukasawa and Mitja Stadje. Perfect hedging under endogenous permanent market impacts. Papers 1702.01385, arXiv.org, February 2017.

[31] Ryan Garvey, Tao Huang, and Fei Wu. Why do traders split orders? *Financial Review*, 52(2):233–258, 2017.

[32] Sahika Genc, Sunil Mallya, Sravan Bodapati, Tao Sun, and Yunzhe Tao. Zero-shot reinforcement learning with deep attention convolutional neural networks. *CoRR*, abs/2001.00605, 2020.

[33] J. C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society, Series B*, pages 148–177, 1979.

[34] Martin D. Gould, Mason A. Porter, Stacy Williams, Mark McDonald, Daniel J. Fenn, and Sam D. Howison. Limit order books. *Quantitative Finance*, 13(11):1709–1742, 2013.

[35] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18:602–610, 2005.

[36] David Griffis. A3c lstm atari with pytorch plus a3g design. `https://github.com/dgriff777/rl_a3c_pytorch`, 2017.

[37] Olivier Guéant. *The Financial Mathematics of Market Liquidity: From Optimal Execution to Market Making.* 03 2016.

[38] Nikolaus Hautsch and Ruihong Huang. Limit order flow, market impact and optimal order sizes: Evidence from nasdaq totalview-itch data. *in:"Market Microstructure: Confronting Many Viewpoints – Conference Proceedings"*, 08 2011.

[39] Dieter Hendricks and Diane Wilcox. A reinforcement learning extension to the almgren-chriss framework for optimal trade execution. In *CIFEr*, pages 457–464. IEEE, 2014.

[40] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. 2017. cite arxiv:1710.02298Comment: Under review as a conference paper at AAAI 2018.

[41] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *CoRR*, abs/1909.00590, 2019.

[42] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[43] Ercüment Ilhan, Jeremy Gow, and Diego Pérez-Liébana. Teaching on a budget in multi-agent deep reinforcement learning. In *CoG*, pages 1–8. IEEE, 2019.

[44] J. Jacob and A. N. Shiryayev. *Limit Theorems for Stochastic Processes.* Springer, 2002.

[45] Sebastian Jaimungal and Damir Kinzebulatov. Optimal execution with a price limiter. *SSRN Electronic Journal*, 01 2013.

[46] Jiongmin Yong and Xun Yu Zhou. Dynamic Programming and HJB Equations. In *Stochastic Controls. Applications of Mathematics (Stochastic Modelling and Applied Probability)*, volume 43, pages 157–215. Springer, New York, NY, 1999.

[47] Jan Kallsen and Johannes Muhle-Karbe. High-resilience limits of block-shaped order books. *SSRN Electronic Journal*, 09 2014.

[48] F.P. Kelly and Elena Yudovina. A markov model of a limit order book: Thresholds, recurrence, and trading strategies. *Mathematics of Operations Research*, 43, 07 2017.

[49] Alexander Kempf, Daniel Mayston, and Pradeep Yadav. Resiliency in limit order book markets: A dynamic view of liquidity. *SSRN Electronic Journal*, 03 2008.

[50] Alexander Kensert, Philip Harrison, and Ola Spjuth. Transfer learning with deep convolutional neural network for classifying cellular morphological changes, 06 2018.

[51] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[52] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks, 2016. cite arxiv:1612.00796.

[53] Charles-Albert Lehalle and S. Laruelle. *Market microstructure in practice.* 01 2013.

[54] Charles-Albert Lehalle and Eyal Neuman. Incorporating signals into optimal trading. *Finance and Stochastics*, 04 2017.

[55] J. Levendovszky and Farhad Kia. Prediction based - high frequency trading on financial time series. *Periodica Polytechnica Electrical Engineering*, 56:29, 01 2012.

[56] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Intell. Res.*, 61:523–562, 2018.

[57] Ananth N Madhavan. Vwap strategies. *Trading*, 2002(1):32–39, 2002.

[58] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

[59] Moritz Voß. *Dynamic Hedging in Illiquid Financial Markets.* PhD thesis, Berlin Institute of Technology, 2017.

[60] Moshe Leshno et al. Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function. *Neural Networks*, 6:861–867, 1993.

[61] Nasdaq. Nasdaq totalview-itch 5.0: Itch is the revolutionary nasdaq outbound protocol.

[62] Yuriy Nevmyvaka, Yi Feng, and Michael J. Kearns. Reinforcement learning for optimized trade execution. In William W. Cohen and Andrew W. Moore, editors, *ICML*, volume 148 of *ACM International Conference Proceeding Series*, pages 673–680. ACM, 2006.

[63] Brian Ning, Franco Ho Ting Ling, and Sebastian Jaimungal. Double deep q-learning for optimal execution. *CoRR*, abs/1812.06600, 2018.

[64] Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Benchmark dataset for mid-price prediction of limit order book data. *CoRR*, abs/1705.03233, 2017.

[65] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *CoRR*, abs/1802.07569, 2018.

[66] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Deep adaptive input normalization for price forecasting using limit order book data. *CoRR*, abs/1902.07892, 2019.

[67] S. Patterson. *Dark Pools: The Rise of the Machine Traders and the Rigging of the U.S. Stock Market.* Crown Business, 2013.

[68] Federico Platania, Pedro Serrano, and Mikel Tapia. Modelling the shape of the limit order book. *Quantitative Finance*, 18:1–23, 02 2018.

[69] Silviu Predoiu, Gennady Shaikhet, and Steven Shreve. Optimal execution in a general one-sided limit-order book. *SIAM J. Financial Math.*, 2:183–212, 01 2011.

[70] Richard Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239, 1958.

[71] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, Massachusetts, 2 edition, 2014.

[72] Robert Almgren and Neil Chriss. Optimal Execution of Portfolio Transactions. 1999.

[73] Ioanid Roşu. A dynamic model of the limit order book. *The Review of Financial Studies*, 22(11):4601–4641, 2009.

[74] Anton D. Rubisov. Statistical arbitrage using limit order book imbalance. 2015.

[75] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. cite arxiv:1609.04747Comment: 12 pages, 6 figures.

[76] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. cite arxiv:1511.05952Comment: Published at ICLR 2016.

[77] Alexander Schied, Torsten Schöneborn, and Michael Tehranchi. Optimal basket liquidation for cara investors is deterministic. *Applied Mathematical Finance*, 17(6):471–489, 2010.

[78] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *ICLR (Poster)*, 2016.

[79] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[80] Alok Sharma, Edwin Vans, Daichi Shigemizu, Keith A Boroevich, and Tatsuhiko Tsunoda. Deepinsight: A methodology to transform a non-image data to an image for convolution neural network architecture. *Scientific reports*, 9(1):1–7, 2019.

[81] Yun Shen, Ruihong Huang, Chang Yan, and Klaus Obermayer. Risk-averse reinforcement learning for algorithmic trading. In *CIFEr*, pages 391–398. IEEE, 2014.

[82] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *J. Big Data*, 6:60, 2019.

[83] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *NeurIPS*, pages 7156–7166, 2018.

[84] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. In Yoshua Bengio and Yann LeCun, editors, *ICLR (Workshop)*, 2015.

[85] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[86] Tom M Mitchell et al. *Machine Learning.* McGraw Hill, 2 edition, 1997.

[87] Lisa Torrey, Trevor Walker, Jude Shavlik, and Richard Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *ECML*, 2005.

[88] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015. cite arxiv:1509.06461Comment: AAAI 2016.

[89] Clara Vega and Christian S. Miller. *Market MicrostructureMarket microstructure*, pages 5392–5404. Springer New York, New York, NY, 2009.

[90] Volodymyr Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. New York, New York, 2016. International Conference on Machine Learning.

[91] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2015. cite arxiv:1511.06581Comment: 15 pages, 5 figures, and 5 tables.

[92] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[93] I. H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data mining : practical machine learning tools and techniques.* Morgan Kaufmann, Amsterdam; London, 2017.

[94] Zhuo Xu, Chen Tang, and Masayoshi Tomizuka. Zero-shot deep reinforcement learning driving policy transfer for autonomous vehicles based on robust control. *CoRR*, abs/1812.03216, 2018.

[95] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *NIPS*, pages 3320–3328, 2014.

[96] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing, 2017. cite arxiv:1708.02709Comment: Added BERT, ELMo, Transformer.

[97] Elia Zarinelli, Michele.treccani Treccani, J. Farmer, and Fabrizio Lillo. Beyond the square root: Evidence for logarithmic dependence of market impact on size and participation rate. *Market Microstructure and Liquidity*, 1, 12 2014.

[98] Zihao Zhang, Stefan Zohren, and Stephen J. Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Trans. Signal Process.*, 67(11):3001–3012, 2019.

[99] Ban Zheng, Eric Moulines, and Frédéric Abergel. Price jump prediction in limit order book. *Journal of Mathematical Finance*, 03, 04 2012.