A MACHINE LEARNING APPROACH TO THE OPTIMAL EXECUTION PROBLEM

by

Quinn Adam Donahoe

B.S. Mathematics, Pennsylvania State University, 2012

B.S. Economics, Pennsylvania State University, 2012

Submitted to the Graduate Faculty of

the Dietrich School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Quinn Adam Donahoe

It was defended on

September 5th 2018

and approved by

Song Yao, Associate Professor (Mathematics)

Michael J. Neilan, Associate Professor (Mathematics)

John M. Chadam, Professor (Mathematics)

Chad J. Zutter, Associate Professor (Business Administration)

Dissertation Director: Song Yao, Associate Professor (Mathematics)

ABSTRACT

A MACHINE LEARNING APPROACH TO THE OPTIMAL EXECUTION PROBLEM

Quinn Adam Donahoe, PhD

University of Pittsburgh, 2018

The Optimal Execution Problem has, for over a decade been of interest in financial mathematics. Solving the problem has, from the mathematics perspective involved using the dynamic programming principle in order to obtain a Hamilton-Jacobi-Bellman PDE to solve for the ideal trading curve. Taking the extended framework of Almgren's 2012 paper on the optimal execution problem with stochastic volatility and liquidity, we begin a statistical learning approach realizing parameters via real market data. From this point, learning algorithms are applied to find optimal trading curves in both limit order and market order strategic environments. We compare these trading curves with trading curves obtained from the classical approach.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

1.0 INTRODUCTION

A good introduction to the problem can be found in both [2] and [3]. We gather information from both sources in our introduction of the problem and the limit order book model.

The optimal execution problem consists of a trader whose portfolio position at time zero contains q_0 shares of a single stock. The goal of the problem is to unwind this portfolio by a given finite time T. By *unwind*, we mean that if $q_0 > 0$, the traders wishes to sell q_0 shares by time T and if $q_0 < 0$, he wishes to buy q_0 shares by time T.

Of course, there is a very easy way to accomplish this with a single market order. For instance, if $q_0 < 0$, the trader could, at any time $t \in [0, T]$ place a market buy order for q_0 shares. Assuming enough liquidity for the stock on the exchange, this order would execute. This simple approach to the problem is fine when q_0 and T are both relatively small.

It is in cases where $|q_0|$ is large that we begin to run into significant problems with this method. In order to see why, we examine the **limit order book**.

1.1 THE LIMIT ORDER BOOK MODEL

We consider a market in which **limit orders** and **market orders** are placed. A market order is an order to buy or sell at the current best price that the market is offering. For example, a trader can place a market order to buy or sell 200 shares of compnay XYZ. This order enters the market, and will execute at the best price for the trader; the lowest price for which 200 shares of XYZ are available. A market order gives the trader a high probability that the desired purchase or sale will take place, but it does not necessarily secure a good price for the trader.

A limit order is different in that it secures a desirable price for the trader, but it does not guarantee that the desired purchase or sale will occur. If the trader wished to obtain 200 shares of the company XYZ via a limit order, he would have to specify not only the number of shares he wished to purchase but also the "limit price" of the order. This price designates the maximum amount that the trader is willing to spend on the shares. This order does not execute until a sufficient opposite side order enters the market. If our trader placed a limit order to purchase 200 shares of XYZ at a price point of 52 dollars, then he would need to wait until a suitable market sell order entered the market to lift his limit order and execute the trade.

When limit orders arrive in an exchange, they are executed on a first in first out basis. So if our trader places a limit order for 200 shares of XYZ at 52$, there is a good chance that there is quite a bit of volume of limit buy orders at that price point as well. So, even if the current best bid price for XYZ is 52 dollars, the trader is not guaranteed of his limit order executing if he is behind in the queue to a large amount of limit order volume. The visualization of how these limit orders sit in the exchange waiting to be executed is known as the *limit order book*.

When a market buy order is sent to an exchange, assuming that there is no corresponding matching market sell order arriving at that exact time, it is routed through that exchange's limit order book. The market order will take all volume of limit orders at the best available price, and then move on to the next best price for the trader. For large values of $|q_0|$ this means that the trader would purchase a significant number of shares at a price much higher ($q_0 < 0$) or much lower ($q_0 > 0$) than the current best market price.

It is for this reason that such a strategy of placing an initial market order for the entirety of q_0 is not ideal. The trader runs a high risk of eating up all of the volume that the market has to offer at the best bid or ask price. We can also condider the opposing extreme strategy, namely, trading q_0 in very small amounts, ensuring that the trader is never buying or selling beyond the best best available price. This could be in fact a very advantageous situation for our trader. For instance, in the case that the trader is purchasing shares, the price may, between time 0 and time T actually fall, and he may over time get better and better best offer prices as time goes on. On the other hand, he is opening himself up to a significant

amount of market risk in that there is a large chance that the price may actually increase in that same time period to the point that he would have been much better off by just walking the book with an initial market order than paying, over time what is a significantly higher price per share.

The optimal execution problem takes note of the fact that both of the extreme ways to solve this problem are suboptimal, so somewhere between minimizing the shielded risk from walking the book (placing small orders over time) and minimizing the risk of price fluctuations due to market risk (instantly placing a large limit order, one finds the actual optimal trading process.

1.2 THE CLASSICAL OPTIMAL EXECUTION PROBLEM

As previously stated, the optimal execution problem asks, for large values of q_0, what is the best way to unwind this portfolio? The problem was first considered in 1998 by Bertsimas and Lo in [4], yet due to the rudimentary nature of this model (it did not account for the change in market price between 0 and T), the credit to the inception of optimal execution research is often given to Almgren and Chriss for their framework given in [5] and [6] a few years afterwards.

The classical mathematical setup is as follows from [2]. We seek an optimal control process v_t where

$$dq_t = v_t dt, \qquad \int_0^T v_t dt = -q_0 \tag{1.1}$$

$$v_t = \underset{v \in \mathcal{A}}{\operatorname{argmax}}\, \mathbb{E}[U(X_T)]. \tag{1.2}$$

Financially speaking v_t represents the trading velocity; how much the trader buys or sells at time t. Thus, the total volume of his trades must unwind his initial position q_0. This is illustrated in (1.1). q_t represents the trader's position at time t and is called the *trading curve*. The trader's cash account process is given by X_t. The optimal control process (1.2) is given to maximize a chosen utility function $U(x)$ amongst the set of admissible strategies $\mathcal{A}$.

A common utility function is the mean-variance critetion: $U(x) = \mathbb{E}[x] - \frac{\gamma}{2}\text{Var}\ (x)$ for some $\gamma > 0$. The key observation here is that the utility function's input is the end time cash process X_T. Someone with a broader understanding of finance may find this odd since it is often the case that one wishes to use their cash account process to purchase stocks with high fundamental values to increase utility. We refer back to the nature of the optimal execution problem in that the fundamental value of the asset itself is irrelevant, and we are only concerned with how we can purchase or sell large volumes of shares of it in the best possible manner for the trader. Thus, his cash account process at time T is where we will derive our utility.

The initial attempts at this problem in [5] and [6] were done in the discrete time setting and had the dynamics as described in [2]:

$$q_{n+1} = q_n + v_{n+1}\Delta t \tag{1.3}$$

$$(v_n)_n \in \mathcal{A} = \left\{ (v_1, ..., v_N) \in \mathbb{R}^N,\ \sum_{n=0}^{N-1} v_{n+1}\Delta t = -q_0 \right\} \tag{1.4}$$

$$X_N = X_0 - \sum_{n=0}^{N-1} v_{n+1} S_n \Delta t - \sum_{n=0}^{N-1} L\left(\frac{v_{n+1}}{V_{n+1}}\right) V_{n+1}\Delta t \tag{1.5}$$

$$S_{n+1} = S_n + \sigma\sqrt{\Delta t}\varepsilon_{n+1} + k v_{n+1}\Delta t, \qquad 0 \le n < N. \tag{1.6}$$

$$U(X_N) = \mathbb{E}[-\exp(-\gamma X_N)],\ \gamma > 0. \tag{1.7}$$

As we can see the time horizon $[0, T]$ has been split into N equally spaced time intervals. In (1.5) the function L is an asymptotically super linear function satisfying $L(0) = 0$ and S_t defines the stock price process for the desired stock. V_t represents the volume of the stock that is being traded by other agents.

In (1.6) we note that $\varepsilon_1, ..., \varepsilon_N$ are independent, identically distributed standard normal random variables. These conditions seek to emulate the random nature of the stock price movement as close to the Brownian motion model as possible in order to retain the traditional structure of the continuous time stock price models while remaining in discrete time.

The solution is found by obtaining the optimal discrete time trading curve $(q_n)_n$ to be

$$q_n^* = q_0 \frac{\sinh(\alpha(T - t_n))}{\sinh(\alpha T)}$$

such that α solves

$$2(\cosh(\alpha\Delta t) - 1) = \frac{\gamma\sigma^2 V}{2\eta}\Delta t^2.$$

The problem was later solved in continuous time by Almgren in [7] under the similar dynamics:

$$\begin{aligned}
\mathcal{A} &= \left\{(v_t)_{t\in[0,T]} \in \mathbb{H}^0(\mathbb{R}, (\mathcal{F}_t)_t),\ \int_0^T |v_t|dt \in L^\infty(\Omega)\right\} \\
dX_t &= -v_t\left(S_t + L\left(\frac{v_t}{V_t}\right)\right)dt \\
dS_t &= \sigma dW_t + kv_t dt.
\end{aligned}$$

The optimal trading velocity was found to be

$$v_t^* = -q_0\sqrt{\frac{\gamma\sigma^2 V}{2\eta}}\frac{\cosh\left(\sqrt{\frac{\gamma\sigma^2 V}{2\eta}}(T-t)\right)}{\sinh\left(\sqrt{\frac{\gamma\sigma^2 V}{2\eta}}T\right)}.$$

The parameter σ represents a constant, arithmetic volatility to the stock. In the above listed models, the volatility is not permitted to change over the time horizon. The parameter k is represents permanent market impact to serve as an indication of liquidity in the market. This realization of liquidity through the lens of price impact is sensible, in that if the trader knows what impact his trades will have on the market, he has a good indication of how much volume is present on his preferred side of the order book. This impact parameter is assumed to be linear so to avoid a pitfall known as *dynamic arbitrage* as defined in [8].

Almgren attempted in a mathematical approach to allow volatility and liquidity to take on a more dynamic nature in [9]. In this model, liquidity (σ_t) and volatility (η_t) were defined as stochastic processes:

$$\eta(t) = \overline{\eta}\exp(\xi(t)) \tag{1.8}$$

$$\sigma(t) = \overline{\sigma}\exp\left(-\frac{\xi(t)}{2}\right) \tag{1.9}$$

$$d\xi = a(\xi)dt + b(\xi)dW_t. \tag{1.10}$$

The problem is solved using the dynamic programming principle (DPP)

$$c(t,x,\xi) = \min_{v} \left[\lambda\sigma^2 x^2 dt + \eta v^2 dt + \mathbb{E}c(t+dt, x+dx, \xi+d\xi)\right],$$

to obtain the Hamilton-Jacobi-Bellman (HJB) Equation

$$0 = c_t + \lambda\sigma^2 x^2 + \min_{v}[\eta v^2 - v c_x] + a c_\xi + \frac{1}{2} b^2 c_{\xi\xi},$$

where $c(t,x,\xi)$ represents the **value function**.

We note that the paper does introduce a model with independent stochastic processes governing both volatility and liquidity, yet a majority of the work in the paper focuses on the above dynamics in which volatility and liquidity exist under the assumption that $\eta\sigma^2$ must be constant. This is a coordinated variation assumption that we would like to lift.

As we moved to lift these assumptions, it became the goal of our research to remove even the stochastic dynamics assumption of volatility and liquidity. Our goal became to model the optimal execution problem from a perspective in which as few assumptions on dynamics had to be made as possible. It is this motivation that inspired the chapters that follow. We allow volatility and liquidity to be as close to their real values as we can by measuring them via their statistical interpretations on real data. From here, we derive a statistical method for the optimal execution problem via learning algorithms.

2.0 BACKROUND

2.1 ARTIFICAL NEURAL NETWORKS (ANN)

2.1.1 Motivation

Artificial Neural networks come from a field of machine learning known as "Deep Learning." We have already stated that our goal is to take input values known to the trader and build an algorithm to predict ideal times to trade. Therefore, we are working with a type of machine learning known as a classification problem.

Standard linear classification models (logistic regression) are fine for models whose inputs do not have complex interactions:

$$\log\left(\frac{p}{1-p}\right) = X \cdot \beta \tag{2.1}$$

$$Y = X \cdot \beta \tag{2.2}$$

For instance, if we had inputs to a model that was originally of the form 2.2, and the goal of this model was to determine how much a golfer's score lowered based on how much they spent on equipment. Thus, the model would seek to find optimal β_0, β_1 such that

$$Y = \beta_0 + \beta_1 * X. \tag{2.3}$$

Where Y is the average score, and X is the price spent on equipment. Such a univariate analysis is not adequate as a golfer's score is a product of far more parameters than the amount they spend on equipment. Therefore we seek to add more variables to the equation. Let's start with skill. The USGA handicap index is a reputable measure of skill for golfers. Now, our model becomes

$$Y = \beta_0 + \beta_1 * X + \beta_2 * H \tag{2.4}$$

where H is the golfer's handicap. This will be a better model, but it's missing something. What it's missing is the knowledge that a golfer's handicap is very much related to his ability to effectively use equipment. The lower the handicap, the better the golfer. The most expensive golf equipment is equipment that is designed so that highly skilled players can control the spin and trajectory of the golf shot to a higher degree of precision. Such shots are advantageous to golfers as they have more options to navigate obstacles and place the ball closer to the hole. This will produce lower scores. A less skilled golfer, one with a high handicap could spend lots of money on equipment, but cannot actually put it to its intended use because he lacks the necessary skill to take advantage of the equipment's features. Due to this being quite common knowledge, it is often the case that skilled players will spend more than unskilled players on equipment, as it is known to improve their games.. Therefore, a more appropriate model recognizes that there is an *interaction* between handicap (H) and price spent on equipment (X) will effect the golfer's score.

$$Y = \beta_0 + \beta_1 * X + \beta_2 * H \beta_3 * X * H \tag{2.5}$$

The simple interaction term of $X * H$ is adquate for the purposes of a model like golf scores. It could show a marketing team how to adequately sell to aggregations of large categories of vaguely similar golfers. This model, is different from the one we are working with in regards to how accurately we wish to explore the interactions. In financial data, more specifically the optimal execution problem, we are concerned with high frequency data. We will have second by second statistical realizations of things such as volatility, liquidity, stock price, and derivative (trend) of the stock price. We know that there are for certain interactions between things like volatility, liquidity, and price movements. However, these aren't simple interactions at the high frequency level. The most simple interaction would likely be volatility as it is often calculated via historical (standard deviation), GARCH, or

implied volatility. However, as we see from each of those relationships, they are hardly as simple as Price*Volatility.

We seek a machine learning model that is able to capture highly complex interactions between variables. The ideal model for this is an artificial neural network, which grew out of a desire to model regression problems with nonlinear structures.

2.1.2 Feed-Forward Sequential Neural Networks with Back-propogation

We will begin by explaining the simplest neural network model that would complete our binary classification task as described in [10]. Neural networks are best understood via their visual representation but we will begin, as a normal binary classification problem would. We are given an input vector X that contains p inputs. The goal is for the model to output

$$\{f_1(X), f_2(X), ..., f_k(X)\}$$

where f_k represents the probability that the input vector X is of class k. A basic neural network mode to carry out such a computation is done as follows:

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X); \qquad m = 1, ..., M. \tag{2.6}$$

$$T_k = \beta_{0k} + \beta_k^T Z; \qquad k = 1, ..., K. \tag{2.7}$$

$$f_k(X) = g_k(T); \qquad k = 1, ..., K. \tag{2.8}$$

Described in (2.6) is what is known as the *hidden layer* of a neural network model. Each Z_m represents one node of the hidden layer. Each node X_i in the input layer interacts with every node in the hidden layer. Consider for instance Z_2.

$$Z_2 = \sigma(\alpha_{02} + \alpha_2^T X).$$

Each line connecting Z_2 to X represents α_{2i} for some $i \in \{1, ..., p\}$. Each vector α_m is therefore a $p - dimensional$ vector (since X is p-dimensional) and the term α_{02} refers specifically to the *bias* term that acts much like the bias term in 2.2. We recall that the main

goal of neural network algorithms is to predict based on nonlinear functions of input. Clearly modeling on only $\alpha_{0m} + \alpha_m^T X$ in the hidden layer would do damage to this purpose. This is where σ comes in. σ is known as the *activation function* in the hidden layer. In original conceptions of neural networks, these activation functions were step functions. However, given the desire for smoothness in the optimization process, the sigmoid activation function was adopted:

$$\sigma(v) = \text{sigmoid}(v) = \frac{1}{1 + e^{-v}}.$$

The sigmoid function is no longer used as widely due to issues we will discuss when going over the specifics of our plan to approach the optimal execution problem in this manner. For now, one can think of the activation function as any smooth nonlinear function, such as sigmoid.

We turn our attention now to (2.7). This layer is calculated much the same as the input to σ in the hidden layer is calculated. For some $k \in \{1, ..., K\}$,

$$T_k = \beta_{0k} + \beta_k^T Z.$$

One thing to note here, is that as in the hidden layer where each α_m was a p-dimensional vector, β_k is an M dimensional vector. This is because each node T_k will be a linear combination of every single element from the hidden layer Z_m, $m = 1, ...M$, where there are M nodes. We once again have the bias term β_{0k}.

After the T_k node is calculated for each k, the model is ready to produce its output. This comes in the form of applying, for each class k the *output function* $g_k(T)$. The most widely accepted output function for classification problems is what is called the "softmax" activation function:

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}. \tag{2.9}$$

We specify here that $g_k(T)$ is the output, and thus $f_k(X) = g_k(T)$ as is stated in 2.8. This represents the prediction of Y_k, the value of the probability that input X is predicted to be

a member of class k. Neural network models are fit to the training data via adjustment of the weights. By weights, we mean the set $\boldsymbol{\theta}$ where

$$\boldsymbol{\theta} = \{\alpha_{0m}, ..., \alpha_m, \beta_{0k}, ..., \beta_k, m = 1, ..., M,\ k = 1, ..., K\}. \tag{2.10}$$

These weights are optimized by minimizing the loss function to the training output data $\{y_i\}_{i=1}^N$. Consider the most common loss function of sum of squared errors:

$$R(\boldsymbol{\theta}) = -\sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} - f_k(x_i))^2. \tag{2.11}$$

Note that here x_i represents a p-vector of observed input data and y_{ik} represents the observed output for class k. If we think of this in the sense of classification problems, this value will always be 0 or 1 because it is clear to the observer which class the observed output belongs to. In classification neural networks these nodes will often have values within the interval $(0, 1)$, and the model tells the user that the tested input is in the class

$$G(x) = \text{argmax}_k f_k(x).$$

We seek to minimize $R(\boldsymbol{\theta})$ with our weights during training, but an approach towards the global minimizer of $R(\boldsymbol{\theta})$ will tend to overfit the model to the training data. Instead we seek other methods of optimization. The most standard optimization technique is known as gradient descent back-propogation.

2.1.3 Gradient Descent Back-Propogation

We consider a technique to best train our data with respect to the loss function of (2.11) from [10]. Assume that our neural network has been forward propogated to create the structure described in (2.6) - (2.8). To observe this back-propogation technique over a single input vector's values, consider the hidden layer for node m of the i-th observed input vector;

$$z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i). \tag{2.12}$$

Consider the loss function for this input vector to be

$$R_i(\boldsymbol{\theta}) = \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2. \tag{2.13}$$

$$\text{From 2.11}, R(\boldsymbol{\theta}) = \sum_{i=1}^{N} R_i(\boldsymbol{\theta}). \tag{2.14}$$

In order to perform an optimization with respect to $R(\boldsymbol{\theta})$, we need to calculate its gradient. For this, we keep in mind the structure of (2.8) and (2.7) to recognize that for a single observation

$$f_k(x_i) = g_k(\beta_{0k} + \beta_k^T z_i). \tag{2.15}$$

We will now proceed with the gradient calculation.

$$\frac{\partial R_i}{\partial \beta_{km}} = \frac{d}{d\beta_{km}} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \tag{2.16}$$

$$= -2(y_{ik} - f_k(x_i) * g'(\beta_{0k} + \beta_k^T z_i) z_{mi}. \tag{2.17}$$

We may also rewrite 2.15 to include the hidden layer so that

$$f_k(x_i) = g_k\left(\beta_{0k} + \beta_k^T(\sigma(\alpha_{om} + \alpha_m^T x_i))\right) \tag{2.18}$$

$$\Rightarrow \frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i) g'(\beta_{0k} + \beta_k^T z_i)\beta_{km}\sigma'(\alpha_{0m} + \alpha_m^T x_i) x_{i\ell}. \tag{2.19}$$

From (2.17) and (2.19), we can now perform what is known as a *gradient descent update* to our weight set. Given the value of the weights at the iterate r, we can obtain the $r+1$ iterate of the weights via

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}} \tag{2.20}$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}}. \tag{2.21}$$

Here, the term γ_r is known as the *learning rate* at iterate r. In order to make the process more computable, we write (2.18) and (2.19) such that

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi} \tag{2.22}$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = s_{mi} x_{il}, \tag{2.23}$$

where δ_{ki} and s_{mi} are observed errors that satisfy the so called "back-propagation" equations

$$s_{mi} = \sigma'(\alpha_{0m} + \alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}. \tag{2.24}$$

The updates to the weights are computed in what is known as a *two pass algorithm*. In the *first pass*, known as the "forward pass" the current weights in $\boldsymbol{\theta}$ are fixed and the predicted output values are calculated via (2.6) - (2.8). That is to say, with current weights, go through the neural network to the output layer. In order to update, we go through the "backward pass." This begins with starting with computing the values δ_{ki} from the output layer using (2.17) in conjunction with the definition of δ_{ki} given in (2.22). From here, we use (2.24) to compute s_{mi}. This allows us to define the values of $\frac{\partial R_i}{\partial \beta_{km}}$ and $\frac{\partial R_i}{\partial \alpha_{m\ell}}$ via (2.22) and (2.23) and finally get the updates to the weights as defined by (2.20)-(2.21).

This procedure is what is defined by the phrase "back-propogation." It is carried out on every piece of input data. One full sweep through the data is termed an "epoch." We may, as practitioners of the algorithm include as many or as few epochs as we wish into a neural network. Often times, more are included until the accuracy of the model begins to converge.

2.1.4 Modernization of the Neural Network for our Model

We now take this basic concept of a neural network and extend it to the nureal network that we will actually use. We will make the following changes to the neural network described in the previous section.

1. We will add more hidden layers.
2. We will specify our activation function σ.
3. We will change the optimizer from gradient descent to an optimization algorithm known as "Adam."
4. We will optimize subject to a different loss function known as "categorical cross-entropy."

Let's first consider the possiblility of adding more hidden layers to the neural network. This is not a difficult extension to envision. Looking to **??**, assume we now have instead of just one hidden layer Z, that we now have H hidden layers $\{Z^1, Z^2, ..., Z^H\}$. These hidden layers are generated much in the same fashion as before. The first hidden layer Z^1 is defined such that

$$Z_m^1 = \sigma_1(\alpha_{0m} + \alpha_m^{1T} X), \qquad m = 1, ...M_1.$$

Note that here α_m^{1T} is a p-dimensional vector like X. This layer Z^1 connects to the layer Z^2 such that

$$Z_m^2 = \sigma_2(\alpha_{0m}^2 + \alpha_m^{2T} Z^1), \qquad m = 1, ...M_2.$$

In this case, we note that the size of the vector α_m^{2T} will differ from that of α_m^{2T} in the sense that α_m^{2T} now has M_1 components in order to properly interact with Z^1. This process continues until the output layer, which is described as

$$T_k = \beta_{0k} + \beta_k^T Z^H$$

where, as expected β_k is a M_H component vector. Thus, we see that it is not too big of an extension on the original framework to add more hidden layers to a neural network. Adding hidden layers can add to the model by extracting more complex features within the data.

We will seek to optimize the number of hidden layers using modern methods for tuning the model as proposed in [11].

Moving on we find the task of choosing the correct activation functions. As we have stated previously in this section, for a while the standard activation function by practitioners was chosen to be the sigmoid activation function

$$\sigma(v) = \frac{1}{1 + e^{-v}}.$$

We know that for optimization purposes, we have to employ the gradients of this function. The gradient of the sigmoid activation function is

$$\sigma'(v) = \frac{e^{-x}}{(1 + e^{-x})^2}.$$

Nothing seems too out of the ordinary here, but imagine using this function for a neural network that has quite a few hidden layers. Here, we reference from our initial walk throug the neural network the function for the hidden layer weight derivative (2.19). As we add more hidden layers to this algorithm, the initial layer Z^1 will have weights such that $\frac{\partial R_i}{\partial \alpha^1_{m\ell}}$ will contain the product quite a few gradients of activation functions. Considering the maximum value of $\sigma'(x)$ in this case is .25, and for values of $v > |5|$ the gradient decays to essentially 0, it isn't advisable to use sigmoid as our activation function since as our gradients vanish, our updates to the weights for these earlier hidden layer nodes (for example the updates defined in (2.20)-(2.21)) will be minimal at best. Therefore, it is not in our best interest to use sigmoid for our activation function nor, for that matter any function whose gradients decay rapidly.

In order to rectify this problem we turn to what is known as the ReLU (rectified linear unit) activation function. It was first used in [12] and was considered according to [13] to be the most popular activation function for deep learning algorithms today. The function is defined such that

$$\text{ReLU}(v) = v^+ = \max\{0, v\}.$$

Thus, the gradient of this activation function is either 0 or 1. This ensures that our gradients do not die out as we add more hidden layers to our neural network model. This

activation function will be used at all of the nodes in our model's hidden layers. For our output layer, as defined by 2.7 and 2.8, we will use the function

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}} \tag{2.25}$$

known as the 'softmax' function. This is the standard output function for classification problems (such as the one we are doing with binary trade tick classification).

We now turn to the idea of optimizing our neural network weights $\boldsymbol{\theta}$ where

$$\boldsymbol{\theta} = \left\{ \alpha^{1,\dots H}_{1,,,M_{h'}}, \beta_{01}, .., \beta_{0k}, \beta_1, .., \beta_K \right\}.$$

For our specific problem of classification, it is better to not use the sum of squares defined loss function and to instead use the loss function known as "categorical cross-entropy" where

$$R(\boldsymbol{\theta}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} f_k(x_i). \tag{2.26}$$

Our goal to optimize this presents us with a couple of options in terms of how we wish to optimize this. We have the example of gradient descent, which can be stated in a more general sense as

$$\theta_j^{(r+1)} = \theta_j^{(r)} - \alpha \frac{\partial}{\partial \theta_j} R(\theta).$$

A weakness of this optimizer is that it generally requires the algorithm to loop over quite a bit of data since it goes through each and every input. In general, deep learning algorithms take quite a bit of data in order to converge on an good model. An extension of this optimizer that rectifies this problem is known as *stochastic* gradient descent (SGD). In this optimizer, we randomly choose an input direction so as to loop over less data in each epoch. In this instance, the update given in (2.20) is changed to

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \mathbf{N} \sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g'(\beta_{0k} + \beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{i\ell}.$$

This method is a step in the right direction towards the optimizer that we wish to use. The optimizer we employ is known as *Adam* introduced in [14]. It is named for its motivation of "adaptive motive estimation." The big difference between Adam and SGD is that SGD, when

implemented maintains a single learning rate γ_r among all weight updates per iterate. Adam gets around this limitation by calculating an exponential moving average of the gradient and squared gradients. It has two user inputted parameters $\boldsymbol{\beta^1}$ and $\boldsymbol{\beta^2}$ that control the decay rates among the moving averages of these gradients. The reason we are employing the Adam optimizer is because of its reputation for good performance. In [15], a lot of different optimization techniques for deep learning neural networks were put to the test. Adam performed the best in this examination, able to beat most methods in the most situations.

Data: Adam$(\alpha, \boldsymbol{\beta^1}, \boldsymbol{\beta^2}, f(\boldsymbol{\theta}), \boldsymbol{\theta}_0)$

Result: $\boldsymbol{\theta_t}$

initialization;

$m_0 = 0,\ v_0 = 0, t = 0$

while $\boldsymbol{\theta_t}$ *is not converged* ($\|\boldsymbol{\theta_t} - \boldsymbol{\theta_{t-1}}\| > tolerance$) **do**

$t \leftarrow t + 1$;

$g_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta_t - 1})$;

$m_t \leftarrow \boldsymbol{\beta^1} * m_{t-1} + (1 - \beta_1) g_t$;

$v_t \leftarrow \boldsymbol{\beta^2} * v_{t-1} + (1 - \beta_2) g_t^2$;

$\hat{m}_t \leftarrow \frac{m_t}{1 - \boldsymbol{\beta^1}}$;

$\hat{v}_t \leftarrow \frac{v_t}{1 - \beta^2}$;

$\boldsymbol{\theta_t} \leftarrow \boldsymbol{\theta_{t-1}} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$;

end

Algorithm 1: The Adam optimization Algorithm

In the above algorithm, the paper puts forward initializing with $\alpha = 0.001, \beta^1 = 0.9, \beta^2 = 0.999$, and $\varepsilon = 10e - 8$. We note that the function f, for the purposes of our problem will be the categorical cross entropy function (loss function) defined with the activation functions of ReLU in the hidden layer and softmax in the output.

2.2 REINFORCEMENT LEARNING

2.2.1 An overview of Reinfocement Learning

In this subsection, a basic overview inspired by [16] is given. In reinforcement learning tasks, according to information learned from [16] , the **agent** interacts with his **envrironment**, and in the various states of his environment, he makes **actions**. As a result of his actions he is given a **reward**.

The structure of a reinforcement learning problem is such that an agent is performing a task in an environment that can be defined by various states. At a set of discrete time steps

$$t = 0, 1, 2, 3, ...$$

the agent receives some representation of the environment's **state**. The state at time t is denoted by S_t:

$$S_t = s \ \in \ \mathcal{S},$$

where $\mathcal{S}$ represents the set of possible states for the environment. A simple example of this is one where the agent is represented by a chess player's control over a single pawn on a chess board. The discrete times t represent an indexing on the moves that a player makes during a chess match. As one who plays chess knows, pawns can move diagonally while attacking, but not otherwise. Therefore it makes sense to model the movements of a pawn based on whether or not they are in position to attack. The state the pawn is in is the set $\mathcal{S}$ such that

$$\mathcal{S} = \{\text{Can Attack, Can't Attack}\}$$

The agent then interacts with the environment and selects an action: A_t where:

$$A_t = a \ \in \ \mathcal{A}(S_t)$$

where $A(S_t)$ represents the set of actions that are available to the agent in the state S_t. In our example of the pawn on a chess board, the action set differs based on state.

$$\mathcal{A}(\text{'Can't Attack'}) = \{\text{Move Forward, Don't Move}\}$$
$$\mathcal{A}(\text{'Can Attack'}) = \{\text{Move Forward, Don't Move, Attack}\}.$$

One step later, as a consequence of the action that the agent took in time t, the agent is given a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$. He, at this time enters a new state, S_{t+1}. The way that he chooses the actions that he takes at each state is called his **policy**. It is given as a mapping from states to probabilities of selecting each possible action. We denote this by π_t. This is denoted by

$$\pi_t(a|s) = \text{ The probability that action 'a' is taken in state 's'.}$$

In our example, we can assume that a reward of 1 is given if the pawn is not killed by our agent's open before our next turn. A reward of -1 is given if the pawn is killed. If our agent is an agressive chess player, we can expect that $\pi_t(\text{'Attack'}|\text{'Can Attack'})$ is close to one.

Most theories presented in reinforcement learning assume that the environment that is being dealt with can be thought of as a Markov Decision Process or an MDP. This means that instead of the path dependent case where the dynamics can only be defined by the complete joint distribution, we only need knowledge of the current state and current action to predict the resulting state and resulting reward probabilites to the same degree of accuracy as one can expect using the full joint distribution. Thus

$$\begin{aligned} p(s', r|s, a) &= \mathbb{P}\left\{S_{t+1} = s',\ R_{t+1} = r\ |S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t\right\}, \\ &= \mathbb{P}\left\{S_{t+1} = s',\ R_{t+1} = r|S_t = s,\ A_t = a\right\}. \end{aligned}$$

The key idea of reinforcement learning is that we use **value functions** in order to structure the search for good policies. These are motivated by the following definition of discounted return of rewards. We define

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned}$$

to be the discounted return at time t for the agent. We note that $0 < \gamma < 1$. This restriction on γ is crucial in the convergence of the algorithm, but also makes sense from an application point of view. We want the actions we take at a given time to be most impacted by rewards given in the more immediate future to his action at time t. This restriction on γ allows the expected rewards at future times to be depressed a bit. This definition allows us to define the optimization for the agent in the sense that

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} |\ S_t = S, \right] \end{aligned}$$

where here, we note that a policy π maps from each state s to the probability of taking action a in state s. This is denoted by $\pi(a|s)$. Therefore, the above value function is the expected reward returned from following policy π starting at state S. Expanding further, we obtain

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} :\ S_t = S, \right] \tag{2.27}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \tag{2.28}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \tag{2.29}$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right], \qquad \forall s \in \mathcal{S}, \tag{2.30}$$

where 2.30 is obtained via definition of expectation in a finite discrete probability space. The equation (2.30) is known as the **Bellman Equation** for v_π. Solving a reinforcement

learning task is equivalent to finding a policy that out-performs all other policies over the long run. We can order policies by saying that

$$\pi' \geq \pi \Leftrightarrow v_{\pi'}(s) \geq v_{\pi}(s), \qquad \forall\, s \in\ \mathcal{S}.$$

Based on the MDP structure, there is always at least one policy that is greater than or equal to all other policies. This is an **optimal policy**, and it is denoted by π_*. All optimal policies share an optimal value function defined by

$$\begin{aligned} v_*(s) &= \max_{\pi} v_{\pi}(s) \\ &= \max_{a} \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s,\ A_t = a] \\ &= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]. \end{aligned}$$

2.2.2 Policy Evaluation

We need to consider how to compute the value function v_π for any policy π. For any value function v, for all $s \in \mathcal{S}$, we have the Bellman equation 2.30 such that

$$v_{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_{\pi}(s')].$$

Therefore if the environment's dynamics are completely known, as in the agent is aware of $p(s', r|s, a)$ then the equation (2.30) becomes a system of linear equations of size $|\mathcal{S}|$, the cardinality of the state space. The unknowns are therefore

$$v_{\pi}(s) :\ s\ \in \mathcal{S}.$$

Therefore, we can employ an iterative numerical scheme. Consider choosing v_0 arbitrarily. By choosing this, we are choosing a value function, mapping $\mathcal{S}$ to $\mathbb{R}$. We use the Bellman equation (2.30) as an update rule.

$$\mathbf{v_{k+1}}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma \mathbf{v_k}(S_{t+1}) | S_t = s] \tag{2.31}$$

$$= \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma \mathbf{v_k}(s')]. \tag{2.32}$$

The magic of using the Bellman rule as an update scheme is that v_π is a fixed point of the algorithm since it follows the Bellman equation so we are assured of equality. In order to write an actual program to handle this, see below:

Data: Input π, the policy to be evaluated., Θ, the tolerance

initialization;

Initialize an array $V(s) = 0$ for all $s \in \mathcal{S}$;

Repeat;

$\Delta = 0$;

for $s \in \mathcal{S}$ **do**

$v \leftarrow V(s)$;

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$;

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

end

if $\Delta < \Theta$ **then**

Output $V \approx v_\pi$

end

Algorithm 2: Iterative Policy Evaluation

Once we have a value function for a policy, we want to know whether or not we should change that policy to deterministically choose an action $a \neq \pi(s) = \text{argmax}_{a \in \mathcal{A}(s)} p(a|s)$. This of course amounts to changing the policy. Therefore, we want to evaluate a change in the policy at a single state to a particular action. It is a natural extension to consider chnages at all states, and to all possible actions, selecting at each state the action that appears to be the best. We call this a **greedy policy** that is defined as $\pi'(s)$:

$$\pi'(s) = \text{argmax}_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s,\ A_t = a] \tag{2.33}$$

$$= \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]. \tag{2.34}$$

We want to update our policies in the following manner: Begin with an initial policy π_0. Then we seek to iteratively practice policy **improvement**:

$$\pi_0 \xrightarrow{\text{Evaluation}} v_{pi_0} \xrightarrow{\text{Improvement}} \pi_1 \xrightarrow{\text{Evaluation}} v_{\pi_1} \xrightarrow{\text{Improvement}} \pi_2 \xrightarrow{\text{Evaluation}} \cdots \xrightarrow{\text{Improvement}} \pi_*.$$

We now have the final algorithm to process this:

1)**Data**: Input $V(s) \in \mathbb{R}$, and $\pi(s) \in \mathcal{A}(s)$, Θ.

2)Policy Evaluation (as before with a slight change);

while $\Delta > \Theta$ *OR* $count < 1$ **do**

$\Delta = 0$;

count = count + 1 **for** $s \in \mathcal{S}$ **do**

$v = V(s)$;

$V(s) = \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$;

$\Delta = \max(\Delta, |v - V(s)|)$

end

end

3)Policy Improvement ;

policy-stable $\leftarrow$ True ;

for $s \in \mathcal{S}$ **do**

old-action $\leftarrow$ True ;

$\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$;

if *old-action* $\neq \pi(s)$ **then**

policy-stable $\leftarrow$ False

end

end

if *policy-stable* $\leftarrow$ *True* **then**

return $V \approx v_*,\ \pi \approx \pi_*$

end

Else: Return to 2

Algorithm 3: Policy Iteration

To conclude, we note that since we are working with a finite Markov Decision Process, this policy improvement process will converge since eventually, as we cycle through the policies in each state, we will arrive at a point where $\pi(s) = \pi'(s)$, and thus the policy will converge.

2.3 LOGISTIC REGRESSION

Logistic Regression is used for our limit order models. We give a brief setup of the model with info from [10]. The logistic regression problem, despite having the word "regression" in the name is actually by nature a classification problem. Given a predictor set X we which to classify such a predictor with a member of a finite classification set

$$\mathcal{G} = \{1, 2, ..., K\}.$$

A standard classification problem, much like the one we saw earlier with the artificial neural network is one of training an algorithm that will make a prediction as to which classifier the input belongs to. Logistic regression works a bit differently in that a logistic regression algorithm will output the *probability* that the input belongs to each class. Therefore, what the logistic regression model outputs is its prediction of the *posterior probabilities*;

$$P(G = K|X = x). \tag{2.35}$$

In order to predict these, the standard logistic regression model uses linear functions of X to act as decision boundaries.

$$\begin{aligned}
\log\left(\frac{P(G=1|X=x)}{P(G=K|X=x)}\right) &= \beta_{10} + \beta_1^T x \\
\log\left(\frac{P(G=2|X=x)}{P(G=K|X=x)}\right) &= \beta_{20} + \beta_2^T x \\
&\vdots \\
\log\left(\frac{P(G=K-1|X=x)}{P(G=K|X=x)}\right) &= \beta_{(K-1)0} + \beta_{K-1}^T x.
\end{aligned}$$

and thus for any given $k \in \mathcal{G}$;

$$\log\left(\frac{P(G=k|X=x)}{P(G=K|X=x)}\right) = \beta_{k0} + \beta_k^T x \tag{2.36}$$

$$P(G=k|X=x) = \exp(\beta_{k0} + \beta_k^T x) P(G=K|X=x) \tag{2.37}$$

$$\Rightarrow P(G=K|X=x) = 1 - \sum_{\ell=1}^{K-1} P(G=K|X=x)\exp(\beta_{\ell 0} + \beta_\ell^T x) \tag{2.38}$$

$$= 1 - P(G=K|X=x)\sum_{\ell=1}^{K-1}\exp(\beta_{\ell 0} + \beta_\ell^T x) \tag{2.39}$$

$$\Rightarrow P(G=K|X=x)\left(1 + \sum_{\ell=1}^{K-1}\exp(\beta_{\ell 0} + \beta_\ell^T x)\right) = 1 \tag{2.40}$$

$$P(G=K|X=x) = \frac{1}{1 + \sum_{\ell=1}^{K-1}\exp(\beta_{\ell 0} + \beta_\ell^T x)} \tag{2.41}$$

$$(2.37) \Rightarrow P(G=k|X=x) = \frac{\exp(\beta_{k0} + \beta_k^T x)}{1 + \sum_{\ell=1}^{K-1}\exp(\beta_{\ell 0} + \beta_\ell^T x)}. \tag{2.42}$$

We note that using the final class K as the denominator for all of the posterior probabilities is arbitrary, and any individual class can be used.

2.3.1 ROC Curve

via [17], the Receiver Operating Characteristic curve or ROC curve is a good technique for examining the performance of our logistic regression algorithm. Consider a binarty logistic regression model suc that $y = 0$ or $y = 1$. Our model will have a "cutoff point" of probability in which the model will predict class $y = 1$. Thus, if our cutoff is .5, then if our model labels $p \geq 0.5$, the input data will be labeled by our model as 1. What happens when we begin to vary this threshold is what the ROC curve indicates.

The ROC curve plots the false positive rate of the model on test data vs the true positive rate as the threshold changes. Therefore when the threshold is equal to one, the model predicts zero for all of the data. Thus, buth true and false positive rates are 0. On the other hand, when the threshold is 0, the model predicts 1 for all input, and thus the true and false positive rates are both 1.

The gauge of how good a model is, is the *area* under the ROC curve. The closer it is to one, the better the model. We will use this metric to gauge the performance of our model.

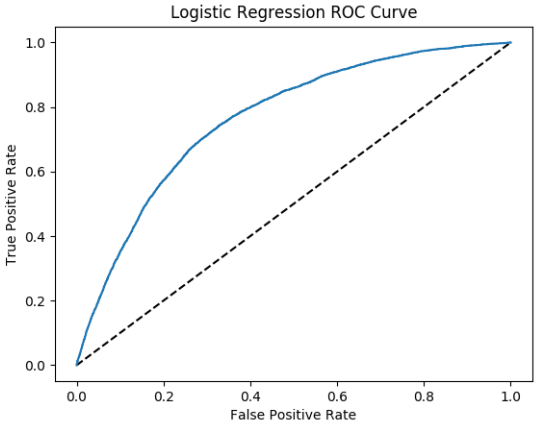

Figure 2.1: The Two different ROC curves are plotted here. The lines represent the ratio of false positives to true positives when the threshold varies.

3.0 OPTIMAL EXECUTION FOR MARKET ORDERS IN LIT MARKETS

3.1 THE INITIAL DATA DRIVEN APPROACH

As discussed in the introduction, one of our goals for this research is to free the models from structural assumptions on things such as volatility and liquidity. Given that volatility (σ) and liquidity (η) are two vital parameters in the classical optimal execution models, it is important for us to statistically realize them for the purposes of our model. We begin with volatility. In later chapters, we have changed ways in which we measure volatility. For this initial data driven approach, we use *historical volatility*. This statistic encompasses the deviation of the price at a current time from some mean value. Here we consider that mean value to be the mean over the past 30 time ticks.

$$\sigma = \sqrt{N} * \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}$$
$$x_i = \log\left(\frac{\text{Close}_{t_i}}{\text{Close}_{t_{i-1}}}\right), \quad \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}.$$

Here, N is the number of periods per year (in our case 390*252), and n is the number of period observations that are used to calculate the average. In our case, the window is set so that the previous 30 observations are used to calculate the average. In statistical terms what we are calculating is the standard deviation of the current log return to its average over the last 30 observations. Thus, after the first 30 ticks in a given stock's data set, we can obtain a volatility statistic based only on the historical close price. Using the previous 30 ticks was our choice. Making this number larger will give the trader a better indication of the mean price, but we must also take note of the fact that there are only 390 minutes of

trading in a given day on the US market. Therefore, we will lose as many data points as we add from the initialization of the model.

Regarding liquidity, we use a statistic known as LIX:

$$\text{LIX} = \frac{1}{10} * \log_{10}\left(\frac{\text{Volume} * \text{Close}}{(\text{High} - \text{Low})}\right). \tag{3.1}$$

The intuition for this statistic is that it takes the total amount of money spent on trading in the market assuming that each trade was for at the close price for the whole interval in the numerator of (3.1) and then divides it by the total amount that the price moved. The intuition behind the statistic as listed in [18] is that

$$10^{10*LIX}$$

is the estimated amount of money needed to create a 1$ price change in the stock price. Thus, for our problem's sake when the LIX is higher, we have a better opportunity to make a big buy without moving the market; so we can make a bigger trade without as much risk of walking the limit order book and losing out on cash for the trader. We thus add 'LIX' to our data set, noting that we have divided it by 10. This serves only to make the value of LIX smaller. We will see once we get to the description of the neural network model that we do not want statistics being too much larger than others or else they can potentially over represent themselves in our limit order book model.

This concludes the data description for the parameters in the model. From these statistics we will begin to make calculations that will lead us toward our trading curve.

3.2 THE CLASSIFICATION LEARNING PROBLEM

In this fist approach we consider the situation of $q_0 < 0$. Thus, the trader must purchase $|q_0|$ shares of the desired equity by time T. A lot of strategies in the classical literature, for instance [19], revolve around a central idea that at the beginning of the execution window, close to time 0, the trader should place a large market order for his desired outcome. In this case, that means that he would place a large market buy order for a significant portion of the

shares $|q_0|$. The idea behind this, is that the presence of a large buyer will excite the market. Therefore, orders to sell will flow in because a large move to the market has occurred. This flow of limit sell orders will happen in a comepetitive manner. That is to say that those placing the limit sell orders will attempt to place them lower than one another to guarantee execution of the sell order. It is believed that from this point on, our trader will be able to keep their eye on the market, and "pick off" these incoming limit sell orders one by one in order to get better prices. In a lit exchange, he is able to observe such trades and incoming limit orders. This will allow him to buy up the stock as it becomes available. At the end of the time horizon, near time T, the trader will place one hopefully small market order for the remainder of $|q_0|$ that he has not yet managed to purchase. The goal is to structure his trades in a manner so that he does not have to place this market order as he has been able to make the entire purchase under the more ideal conditions of picking up market orders.

Our strategy is different from the beginning. This is not a result of us immediately deviating from the classical literature. Our research instead had the goal from the very beginning of using learning algorithms to do this problem. The idea that we developed is an algorithm that can train itself on historical data, and then pick out when the market conditions indicate that it is time to place a market order. From this perspective, the idea of placing a large market order to initialize the strategy was not ideal in terms of the data that we had to work with, that would serve as the basis for training our model.

The data that we are working with is data that was obtained from real markets. The model that hinges on placing this initial large market order is optimized based on knowledge of the type of price impact that the trader's orders have. If we were to assume that we placed a market order at the beginning of our time horizon so large that it moved the market, we would be cheapening the product of the rest of our data as time moved forward. This is because an order so large should in fact have far reaching consequences on the market from volatility, to liquidity, to trading volume, to prices and trades. For example, considering our imported data, if someone placed a considerably large market buy order for McDonalds' during time 62; the price would definitely move, and the maximum price that the stock traded on during the minute would be higher. Hence a higher price given for 'High'. Simultaneously, this trade would eat up all of the volume in the limit order book at lower prices. Thus the

open, and likely close prices for the stock at subsequent times would be higher. Since there has been a change in price, this is going to have an effect on the historical volatility which reflects deviations from a lognormal mean price. The 'Volume' statistic has been rendered completely inaccurate as it does not reflect a large trade that we allegedly placed at the time. Since volume plays a large role in our definition of LIX, this statistic too will be changed by such a trade.

Observing how all of our vital statistics will be effected by a large market order, and taking into account that a machine learning algorithm's effectiveness will depends largely on the accuracy of the data that it is trained on, we decided to forego the approach of an initial large market order followed by subsequent ideal smaller ones.

Intead, we made our algorithm suited to the following goal: find times with optimal market conditions to make a sizable market buy order. It should be noted in light of the previous discussion, that by sizable market order we mean one that will not move the market up very much if at all. We have the LIX statistic at our disposal which indicates how much we will trade. In a post decimalization world, we would like to preserve the integrity of our data by not purchasing more than what the LIX indicates will move the market by more than one cent. Thus, a 'large' market order is an order defined to be the maximum amount that LIX indicates can be traded and not move the market more than one cent.

When choosing ideal times to trade we decide that we are looking for the following characteristics in a time tick.

1. The **Close Price** must be a minimum of the surrounding ($\pm$) 10 ticks.
2. The **Volatility** is less than the mean volatility for that day.
3. The **Liquidity** is greater than the mean liquidity for that day.

We of course note the reality that this criteria is impossible to check in real time. After all, the trader does not have access to future information. Therefore, he cannot tell that the stock price is at a local minimum, nor can he tell when the volatility or liquidity is truly lower than the mean of that day since the full day hasn't been played out yet. For now, all that we are doing is labeling the data so that we can effectively train our algorithm to locate what we are looking for. At this point, we have arrived at our first machine learning

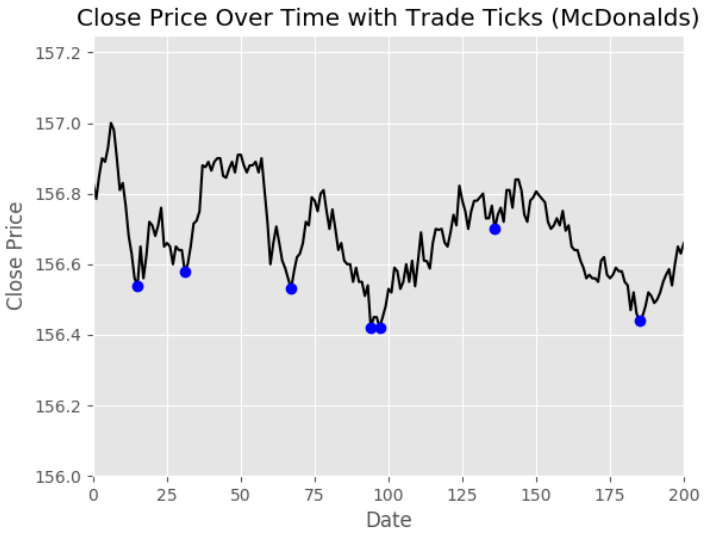

Figure 3.1: The TradeTicks are labeled on the graph above as blue dots.

task; namely a classification learning problem. Its task will be to take as input, data that the trader has access to at a given time, and decide whether or not it is smart, at that time to make a trade. Therefore, the first step that we must take in solving this problem is to effectively label the data. This is shown in in Figure 3.1. We note that in doing this, the trader was allowed to look at the historical data going both backwards and forwards in time. He will use this historical information to train his model, but that model, when used will not allow him to use the same inputs used to classify the data as a '**TradeTick**' (a time in which to place a large market order). Now, our goal becomes to use data from **only previous ticks**, to predict whether or not the current tick is a TradeTick. In order to do this we employ an artificial neural network (ANN).

Date	Close	Open	Volume	LIX	Volatility	**TradeTick**
62	157.72	157.73	3700	.692	.152	**1**
63	157.76	157.73	13118	.747	.152	**0**

Figure 3.2: A glimpse of the statistics for the initial data driven approach.

Algorithm	
Inputs (Predictors)	Output (Predicted)
OFR at $t-1, t-2, ..., t-5$	TradeTick
OFRSIZ at $t-1, t-2, ..., t-5$	0 or 1
Volatility at $t-1, t-2, ..., t-3$	
BID at $t-1$	
BIDSIZ at $t-1$	

Figure 3.3: A neural network structure for the initial data driven approach.

3.3 NEURAL NETWORK MODEL FOR OPTIMAL EXECUTION

As seen in Figure 3.3, 15 total input nodes will predict whether our current tick is one to trade at. This information is all available to the trader before he decides whether or not to make a trade based on that tick. Our neural network model is made using keras [20].

Our algorithm takes the previous 20 ticks and predicts whether or not a trade should be made at the current tick. Recall that the main goal is to determine how to purchase q_0 shares of stock within a finite time horizon $[0, T]$. So far, we only know when the ideal points are to make our trades. We still need to know how much to trade at each tick. This is accomplished via our reinforcement learning algorithm.

3.3.1 Reinforcement Learning Algorithm for Optimal Execution

A reinforcement learning algorithm will be used to determine how many shares to purchase at each **trade tick**. We are limited, in that this requires us to transfer our problem into a Markov Decision Process.

We note that we need the following completely defined:

1. $\mathcal{S}$
2. $\mathcal{A}(s)$
3. $\mathcal{R}$
4. $p(s', r|s, a)$

We begin with the state space $\mathcal{S}$. We wish to define the state space based upon two things: Amount left to trade, and time of the day. In a perfect world with infinite computing power, we could make every minue of the day a possible state, and every dollar amount a possible state. This is not within the confines of any reasonable computation. This is why we have split the states into the following

$$\mathcal{S} = \{\mathbb{X}, \mathbf{T}\} \tag{3.2}$$

$$\mathbb{X} = \{X_1, X_2\} \tag{3.3}$$

$$\mathbf{T} = \{t_1, t_2, t_3\} \tag{3.4}$$

- t_1: Represents the time between the minutes [1,130)
- t_2: Represents the time between the minutes [130,260)
- t_3: Represents the time between the minutes [260,390)
- X_1: Represents that there is between X and $\frac{1}{2}X$ left to be purchased where X is an inputted share amount that the trader desires to purchase
- X_2: Represents that there is between $\frac{1}{2}X$ and 0 left to be purchased where X is an inputted share amount that the trader desires to purchase

Therefore, we can now fully define $\mathcal{S}$, a set of cardinality 7. Recall, that this cardinality plays majorly into the computation of our ideal value and policy function.

$$\mathcal{S} = \{(X_1, t_1), (X_1, t_2), (X_1, t_3), (X_2, t_1), (X_2, t_2), (X_2, t_3), \} \tag{3.5}$$

$$\mathcal{S}^+ = \{(X_1, t_1), (X_1, t_2), (X_1, t_3), (X_2, t_1), (X_2, t_2), (X_2, t_3), \text{TERMINAL}\} \tag{3.6}$$

The TERMINAL state occurs if we have completed the optimal purchase amount.

From each state in the state space, our trader will perform an action. He selects his action from the action set. Each action set is listed below

$$\text{for } s \in \mathcal{S}\ \mathcal{A}(s) = \{\text{Not Buy, Buy Half, Buy All}\}$$
$$\text{for } s = \text{TERMINAL}\ ,\ \mathcal{A}(s) = \{\text{Not Buy}\}\,.$$

We designate these as follows:

1. Not Buy: The agent does not purchase any shares
2. Buy Half: The agent buys half of the shares available at the best LOB level.
3. Buy All: The agent buys all of the shares available at the best LOB level.

As a result of his action in the current state the agent is given a reward. The reward set is as follows:

$$\mathcal{R} = \{r_{\text{match}} = +3,\ r_{\text{slip}} = -1,\ r_{\text{shock}} = -10\}\,. \tag{3.7}$$

- r_{match}: This is the reward that our trader is given if his trade matches the LOB value that he intended. This is the desired outcome for the trader.

- r_{slip}: This is the reward that our trader is given if his trade walks the book. Thus, he is given a negative reward if he doesn't get the best price for his intended trade.
- r_{shock}: Sometimes, the market can go crazy. A fire burns down all of the equity's factories. A country commits an act of war, etc. The market can move horrifically in these siutations, and the trader can fall victim. This is a massive negative reward representing a market shock.

We now need to complete this Markov Decision Process. This means that we need to assign all values to the probabilities $p(s', r|s, a)$. The ij-th entry of the matrix is the probability that state-action pair i puts the trader in state s' with reward r as stored in j. The storage of state action and state reward pairs is done manually.

At this point the transition proabilities assigned above have been inputted based on intuition. This will change with the advent of the new supervised learning model that we will implement to learn both the size of the matrix and the transition probabilities.

The intuited proabilities, as well as this initial data driven approach described in this section delivered vastly inferior results to future approaches. They are worth re-iterating in order to see what the framework of our solution began with at the proposal phase, and what is morphed into in the sections that follow.

4.0 HAWKES PROCESS APPROACH FOR MARKET ORDERS IN LIT MARKETS

The TAQ approach is named for its use of TAQ data from the WRDS website. WRDS is a financial database that the University of Pittsburgh subscribes to. It is maintained by the University of Pennsylvania. The TAQ data is called TAQ data as it refers to data of "Trades" and "Quotes." We will use this data to build a limit order book model, build a reinforcement transition matrix, and to build a neural network to find optimal trade times.

4.1 LIMIT ORDER BOOK MODEL AND PARAMETER REALIZATIONS

The first step in using the TAQ data is to build an adequate estimation model of the limit order book at each second of the trading day. The ideal limit order book data would give us, the trader the following knowledge at every tick:

1. The accurate spread between the bid and ask side of the limit order book.
2. The exact volume of limit orders at each price point on both the bid and ask side.
3. The volume should include not only the volume in the exchange, but the volume aggregated amongst all exchanges that the exchange routes orders to. This is because certain laws in [21] require orders to be re-routed in order to ensure that the consumer of the market is given the best possible price. Thus, we would want to see this other volume in a limit order book model because it is volume that will be given to the trader placing a market order.
4. We would want to watch the prices and volume move in continuous time.

We lack access to such a perfect picture of the data we desire, so we build a model for it.

For the mid-price, we have a perfect snap-shot of the limit order book at each time as a result of taking the weighted average of the quoted bid and offer price as well as the volume listed there at each side. A simplified description of the limit order book model that we can glean from the data is that we have the exact weighted average picture across each second that the market that is limited only to the bid price, ask price, and volume available at each tick. This is lacking in regards to the picture that we desire of a complete view of the limit order book. We are lacking any data on liquidity from anywhere beyond the best bid and offer prices. This may seem like quite a bit to lose out on, but in perspective of most HJB models using a block shaped limit order book throughout the entire duration of the model, we are doing a bit better (than most).

We also forego the LIX statistic here in favor of realizing the liquidity of the equity as seen from simply the volume available at each bid and offer price. This is in the spirit of the LIX which gave the trader an indication of how much volume needed to be traded to move the market by one dollar. With the knowledge of the amount of volume at each side of the book, this concept is much more illuminated, in that we know exactly how much one can purchase or sell at the market without moving the market by a tick.

So far, the approach to the TAQ data has been rather similar to our initial data approach, in that we gather some easy to observe statistics from the market and insert them into a machine learning model. We make a large deviation from that with respect to how we measure volatility.

4.1.1 Hawkes Process Model for Microstructure Noise

In the previous section, we realized volatility through the lens of historical volatility. As we said in the previous section this meant that we were measuring the standard deviation from the mean over the last 30 ticks in a moving window. We should note that this method, is generally annualized by the number of trading days per year, not using our method which involved annualizing it via the number of trading minutes in a given year. The reason for this, which encompasses the weakness of measuring the volatility this way, is that when the time scale gets too small, the standard deviation model for historical volatility ends up measuring little more than microstructure noise.

Since microstructure noise is random and complicated, if we are to model it, then we want to do so with a model equipped to handle it.

A modified GARCH model for the purpose of forecasting was considered, as was measuring implied volatility. Similar to the issues with historical volatility, we found that GARCH(1,1), which only looks back by one tick, was not a good model for such small time scales. Extending it back further by means of GARCH(n,m) did not improve the accuracy of the volatility forecast, and neither did modifications on the GARCH forecasting for smaller time scales. Implied volatility was not an option for the reason that the TAQ database for option prices was unavailable to us for all months of the year other than February 2014, while we had TAQ only up to 2011. Therefore, we could not find option prices that adequately reflected the data for which we had stock quotes.

In order to solve the problem of measuring intraday volatility for our model, we turned instead to a measure of *realized volatility*. Realized volatility (often called the signature plot) is defined for an equity over a time period $[0, T]$ with scale τ such that

$$\hat{C}(\tau) = \frac{1}{T} \sum_{n=0}^{\frac{T}{\tau}} \left[X((n+1)\tau) - X(n\tau) \right]^2 . \tag{4.1}$$

As Almgren points out in [22], a typical application of this is to assign the volatility for $T = 1$ day and $\tau = 5$ minutes. This gives a daily number that realizes intraday volatility for that day. This is different from our earlier model where volatility was changing minute by minute on each day. However, as we will see, this number being constant on the day is not

by any means an assumption of constant volatility. The realized volatility itself will change every day, and by training MLE parameters for the prediction of this value, we will arrive at a much more rich model for what we sought with the volatility statistic, namely a predictive model as to when the stock price is expected to jump up or down.

In order to make a predictive model for 4.1, Bacry and others in [23] employed the use of Hawkes Processes. Hawkes processes have been around since the 1970s by A.G. Hawkes in [24] for research on earthquakes. The motivation for Hawkes processes in finance is representative of the idea of modeling the flow of orders and price movements via the Poisson counting process idea. Consider the standard Poisson counting process where in the interval $[0, \delta]$, the arrival of an event such as a market order entering the market happens at the random rate of λ. Then the expected number of arrivals of market orders in this equity for the interval $[0, \delta]$ is given by the poisson random variable X with distribution Poisson$(\lambda\delta)$ and thus it is expected that $\mathbb{E}[X] = \lambda\delta$ orders will arrive.

This is a standard modeling technique for order flow that is often found in [3]. What a Hawkes process does is incorcorate another layer to the model, namely, the idea of a "self-exciting" Poisson process. In such a process, the intensity λ is no longer constant. This is an advantage because in market microstructure, the rate of arrival of things such as order flow and price movements will not remain constant. If we consider for instance, an event throughout the trading day that increases demand for a stock, then this will be realized via the arrival of more buy orders for the stock. Therefore, a sharp increase in arrival of buy orders is associated with an *increase in the expected number of orders to arrive per unit time.* Therefore, the Poisson process's inensity should increase, not remain the same. Consider an elementary Hawkes process example below.

In this Hawkes model, N_t is a point process (Poisson arrival process) with intensity λ_t where λ_t is defined as

$$\lambda_t = \lambda_0 + \int \phi(t-s)dN_s \tag{4.2}$$

$$= \lambda_0 + \sum_{t_i<t} \phi(t-t_i). \tag{4.3}$$

where t_i represent the arrival times of the point process N up to time t, and λ_0 represents a base intensity. If $\phi(x) = \alpha e^{-\beta x}$ where $\alpha\ \beta > 0$, then 4.2 becomes

$$\lambda_t = \lambda_0 + \sum_{t_i<t} \alpha e^{-\beta(t-t_i)},$$

Now, we see that since $\beta > 0$, λ_t is larger depending on how near t is to a concentration of arrival times, and depending on the size of β, decays back closer to λ_0 the further one gets away from a bunch of arrival times. The idea is that in this model, arrivals now imply more arrivals in the near future.

The model for predicting 4.1 is defined by Bacry in [23] as follows. Consider the midprice of a stock X_t that is defined by

$$X_t = X_0 + N_t^1 - N_t^2. \tag{4.4}$$

Here, X_0 represents the stock price at time 0, N_t^1 the arrival process of *upward* price movements of \$.01, and similarly N_t^2 is an arrival process for downward price movements of \$.01. The intensities for these point processes are given by

$$\lambda_t^1 = \mu + \int_{-\infty}^{t} \phi(t-s) dN_2(s). \tag{4.5}$$

$$\lambda_t^2 = \mu + \int_{-\infty}^{t} \phi(t-s) dN_1(s), \tag{4.6}$$

where $\phi(x) = \alpha e^{-\beta x} \mathbb{1}_{\mathbb{R}^+}(x)$, $\alpha, \beta > 0$ and

$$\|\phi\|_1 = \frac{\alpha}{\beta} < 1.$$

A key observation from 4.5 and 4.6 is that they are not self-exciting, but instead *mutually exciting*. What we mean by this is that the jumps near time t in N^1 will cause a stronger intensity for N^2 and vice-versa. Note that in the description above for Hawkes processes, we were talking about market order arrivals. In this coupled Hawkes process, there is mutual excitement instead of self excitement because of the mean reversion principle. This principle of market microstructure states that over time, stock prices will tend towards their mean price. This means, that if the model is showing more jumps up in price, it must have a larger propensity to jump back down.

The model given in [23] showed that the realized volatility $\hat{C}(\tau)$ can be predicted via the following.

Theorem 4.1.1. *If* $\|\phi\|_1 < 1$, *then*

$$C(\tau) = \Lambda\left(\kappa^2 + (1-\kappa^2)\frac{1-e^{-\gamma}}{\gamma\tau}\right) \tag{4.7}$$

$$\Lambda = \frac{2\mu}{1-\|\phi\|_1} \tag{4.8}$$

$$\kappa = \frac{1}{1+\|\phi\|_1} \tag{4.9}$$

$$\gamma = \alpha + \beta. \tag{4.10}$$

The TAQ data gives us a snap shot into what the jumps up and down of the midprice are, as well as a very good realization of $\hat{C}(\tau)$. Therefore, using maximum likelihood estimation on $\Phi = \{\mu, \alpha, \beta\}$, we have the capabilities to predict, based on the TAQ data the forecasted realized volatility, as well as the parameters that will allow for a simulation of the upward and downward jumps of the stock price. This gives our model an measure of intraday volatility via 4.7 as well as a forecast of when to expect upward and downward jumps in the stock price via the Hawkes process for X_t that can now be simulated.

4.1.2 Maximum Likelihood Estimation

Aside from 4.1.1, we also, given the full time series data for movements of stock across the entire day, we know the exact arrival times for the up and down movements of the midprice. Therefore, instead of using some regression technique that will fit the parameters $\boldsymbol{\Theta} = \{\mu, \alpha, \beta\}$ to the realized volatility as obtained by 4.1.1 in 4.7, we actually have a complete snap short of the Hawkes processes N^1, and N^2, and we can use maximum likelihood estimation to fit the parameters to the movements of the midprice.

This requires the use of the log likelihood function which is given for a stationary Poisson process with intensity $\lambda_{\boldsymbol{\Theta}}$ in [23] as

$$L(\boldsymbol{\Theta}) = \int_0^T \log(\lambda_{\boldsymbol{\Theta}}(t))dN_t + \int_0^T (1-\lambda_{\boldsymbol{\Theta}}(t))dt. \tag{4.11}$$

For our situation where our Hawkes intensities with a fixed set of parameters $\mathbf{\Theta} = \{\mu, \alpha, \beta\}$ are given by 4.5 and 4.6, we see that 4.11 becomes (for 4.5)

$$L_1(\mathbf{\Theta}) = \int_0^T \log(\lambda^1_{\mathbf{\Theta}}(t))dN_t + \int_0^T (1 - \lambda^1_{\mathbf{\Theta}}(t))dt \tag{4.12}$$

$$= \int_0^T \log\left(\mu + \sum_{t_j^{(2)}<t} \alpha e^{-\beta(t-t_j^{(2)})}\right) dN_t^1 + \int_0^T \left(1 - \left(\mu + \sum_{t_j^{(2)}<t} \alpha e^{-\beta(t-t_j^{(2)})}\right)\right) dt \tag{4.13}$$

$$= \sum_{t_i^{(1)}<T} \log\left(\mu + \sum_{t_j^{(2)}<T} \alpha e^{-\beta(t_i^{(1)}-t_j^{(2)})}\right) + T(1-\mu) - \sum_{t_j^{(2)}<T} \frac{\alpha}{\beta}\left(1 - e^{-\beta(T-t_j^{(2)})}\right). \tag{4.14}$$

In 4.14, $\{t_i^{(1)}\}$ and $\{t_j^{(2)}\}$ represent arrival times of the events in N^1 and N^2 respectively. That is, they define the times of upward and downward jumps of our price process. Similarly, for N^2, we can calculate from 4.11 that

$$L_2(\mathbf{\Theta}) = \sum_{t_i^{(2)}<T} \log\left(\mu + \sum_{t_j^{(1)}<T} \alpha e^{-\beta(t_i^{(2)}-t_j^{(1)})}\right) + T(1-\mu) - \sum_{t_j^{(1)}<T} \frac{\alpha}{\beta}\left(1 - e^{-\beta(T-t_j^{(1)})}\right). \tag{4.15}$$

Now, given that we have a lot of training data for $\{t_i^{(1)}\}$ and $\{t_j^{(2)}\}$, we can use 4.14 and 4.15 to do maximum likelihood estimation on the compound process via finding $\hat{\mathbf{\Theta}}_{\text{MLE}}$ such that

$$\hat{\mathbf{\Theta}}_{\text{MLE}} = \text{argmin}_{\mathbf{\Theta}} - ((L_1(\mathbf{\Theta}) + L_2(\mathbf{\Theta}))). \tag{4.16}$$

Once here, we can simulate the Hawkes process for microstructure noise directly.

4.1.3 Simulation of the Hawkes Process

In order to simulate the Hawkes Process we follow Bacry in [23] and turn to a so-called *thinning* algorithm proposed by Ogata in [25]. This algorithm can be modified to model for Hawkes processes modeling arrival times once we have found $\hat{\boldsymbol{\Theta}}_{\text{MLE}}$. It rests on the central proposition from the paper:

Proposition 4.1.1 (Ogata 1981 [25])**.** Consider a multivariate point process $(N^p, F, P),\ p = 1, 2, ..., m$ on an interval $(0, T]$ with joint intensity $(\lambda, F) = \{\lambda_t^p, F_t\},\ p = 1, ..., m$. Suppose we can find a one dimensional $F-$predictable process λ_t^* which is defined pathwise satisfying

$$\sum_{p=1}^{m} \lambda^p \leq \lambda_t^*, \qquad 0 < t \leq T,$$

P-almost surely and set

$$\lambda_t^0 = \lambda_t^* - \sum_{p=1}^{m} \lambda_t^p.$$

Let $t_1^*, t_2^*, ..., t_N^* \in (0, T]$ be the points of the process (N^*, F, P) with intensity process λ_t^*. For each of the points, attach a mark $p = 0, 1, ..., m$ with probability $\frac{\lambda_{t_j^*}^p}{\lambda_{t_j^*}^*}$. Then the points with marks $p = 1, 2, ..., m$, provide a multivariate point process which is the same as that given above.

We can then modify 4.1.1 from the algorithm for the simulation given in [25] for the purposes of the Hawkes process that we are dealing with. This is given as Algorithm 4.

1)**Data**: Input $\hat{\mathbf{\Theta}}_{\text{MLE}}$

2)Initialize:;

$N^1 = \{\}$;

$N^2 = \{\}$;

3)Iteration: Below, $\mathcal{U}$ represents the uniform distribution.;

while $s \leq T$ **do**

$\bar{\lambda} = 2\mu + \sum_{n=1}^{2} \sum_{t^* \in N^n} \alpha e^{-\beta(s-t^*)}$;

$u \sim \mathcal{U}_{[0,1]}$;

$w = \frac{\log(u)}{\bar{\lambda}}$;

$s = s + w$ (s is the new time point candidate);

$D \sim \mathcal{U}_{[0,1]}$;

if $D\bar{\lambda} \leq \sum_{n=1}^{2} \lambda^n(s) = 2\mu + \sum_{n=1}^{2} \sum_{t^* \in N^n} \alpha e^{-\beta(s-t^*)}$ **then**

Find $k \in \{1, 2\}$ such that $\sum_{m=1}^{k-1} \lambda^m(s) < D\bar{\lambda} \leq \sum_{m=1}^{k} \lambda^m(s)$;

Assign candidate s to dimension k if $s \leq T$:;

$N^k = N^k + s$

end

end

Algorithm 4: Thinning Hawkes Simulation Algorithm (Ogata) via [1].

4.2 THE NEURAL NETWORK MODEL FOR LOCATING IDEAL TRADE TIMES

As in the initial data driven approach, the goal of this technique is two-fold, to highlight the best times at which to place market orders, and to decide for how much the market orders should be for. In our previous neural network model, the hope was that by looking only at past price ticks, that the neural network model could pick up on nuances in the market and identify future local minimum values in the microstructure noise. We could then predict future minimum values of the price. This proved to be a very lofty goal, and one that didn't quite come to form as we had hoped. Now, armed with the TAQ data, we are able to better our model, and simplify our neural network.

Similar to the initial data driven approach, the goal of this learning method applied via the ANN is to classify what we call 'TRADETICKs', using only past data. Consider our old criteria for classifying the TRADETICKs:

1. The **Close Price** must be a minimum of the surrounding ($\pm$) 10 ticks.
2. The **Volatility** is less than the mean volatility for that day.
3. The **Liquidity** is greather than the mean liquidity for that day.

This criteria gave our learning model a disadvantage from the very beginning. Based purely on financial intuition, it is not all that common for stocks with low volatility to have such dramatic dips in their stock price. Therefore, when whatever measure of intraday volatility is low, there isn't likely to be a valley that implies a minimum across a twenty tick span. Also it has been widely believed since Stoll's work in 1978 in [26], that high volatility coincided with large amounts of illiquidity. This means that when the price is moving to the point where it creates a minimum across 20 ticks, it is difficult to have a low measure of intraday volatility, which implies a low measure of liquidity at that time. What this difficult criteria did was create quite a few number of ticks throughout the day being classified as what we would call TRADETICKs.

This leads to a problem in the machine learning vernacular known as "class imbalance." When a model has class imbalance, it will often return a high degree of accuracy by simply

learning that can mark every single node as one class. For instance, with our model, this would often mean marking every single point in a class of 0 or 'not a TRADETICK.' One can try to remedy this by making the model treat every single marked TRADETICK as many more instances of the model itself. This, though it fixes the problem of class imbalance, gives the machine so many copies of a single time's data that it often leads to the model over-fitting.

Thus, when designing our new neural network model, we need to keep such things in mind. Therefore we seek the following criteria for an ideal trading tick:

1. The **MidPrice** must be among the bottom K percent of mid-prices.
2. The **Liquidity**, as realized by 'OFRSIZ' is greather than a user inputted liquidity threshold.

This is a significant departure from our previous criteria. First take note that we are using MidPrice. The reason that we are doing this is that the way in which we are forecasting microstructure noise is with the Hawkes model as indicated above. This model is designed to predict movements in the midprice. Therefore, we sacrifice an accuracy that may exist in working only with the offer price, but so long as the bid and ask prices for the stock remain consistently close to one another, we see no problem in modeling on the midprice.

Secondly, we see that volatility has been taken out of the ANN model completely. This is for the simple reason that our measure of volatility is now one of realized volatility that produces a metric of intraday volatility for the entire day instead of a unique measurement for every tick. The realized volatility will appear in our ANN model as an input parameter but will not be a criteria for identifying trading times. If was some type of threshold, our model would not return any ideal trading times for days when the realized volatility was forecasted to be too high.

As a note on liquidity, we want liquidity to by relatively high so that the trader can comfortably place a large market order, but it is the understood that we will not impose too high of a burden on liquidity. In doing elementary data analysis, the median seemed to be a reasonable benchmark. This can be changed to be lessened a bit if the user of the algorithm wishes to create less class imbalance. One should note however, that if the agent wishes

Algorithm	
Inputs (Predictors)	Output (Predicted)
-Time t in the trading day in the interval $[0, T]$	TradeTick
-$\mathcal{O}_t$ (Offer Volume)	0 or 1
-$\mathcal{RM}_t$	
-$\mathcal{RMD}_t$	
-$\mathcal{OD}_t$	
-$\mathcal{H}_t$	
-$RV_{[0,T]}$	

Table 4.1: Input parameters and desired output for the artificial neural network model to predict optimal trading times.

to impose the threshold to be a numerical threshold instead of a global descriptive statistic such as the median (i.e. there are at least 300 shares available at the best offer price), then there is no actual need to place liquidity parameters into the neural network since liquidity can be observed on a per second basis.

We now introduce the ANN model that we will use to classify these optimal trading ticks. A table of the paramers is given in 4.2.

We measure the time from the beginning of the horizon that the trader has to make a trade and include it as a parameter in the ANN as $t \in [0, T]$. This parameter is most useful when the time interval is rather uniform across all of the trials of our model. What we mean by this is that this statistic will help our algorithm more if time 0 is always the beginning of the trading day, and time T is the close of the market. This is because trading activity for any given equity has patterns of activity, with the most common being a large volume of trades that occur at the beginning or the end of the trading day. This time index can help our neural network model to recognize relationships between the time of the day and other variables.

We include as $\mathcal{O}_t$ a continuous variable at time t defined as the average best offer volume in the limit order book over the past five seconds.

$$\mathcal{O}_t = \frac{\text{`OFRSIZ'}_t + \text{`OFRSIZ'}_{t-1} + \cdots + \text{`OFRSIZ'}_{t-4}}{5} \tag{4.17}$$

Also included as a parameter in the ANN model is a boolean variable $\mathcal{OD}_T$, a boolean variable that indicates whether or not the best offer volume is decreasing at time t.

$$\mathcal{OD}_t = \begin{cases} 1, & \text{`OFRSIZ'}_t - \text{`OFRSIZ'}_{t-1} < 0 \\ 0, & \text{`OFRSIZ'}_t - \text{`OFRSIZ'}_{t-1} \geq 0. \end{cases} \tag{4.18}$$

We include the variables $\mathcal{O}_t$ and $\mathcal{OD}_t$ in order to give the ANN a signal of liquidity. The structure of this was decided upon after doing a lot of data analysis on tick by tick movements in the best offer size volume. We noticed that over the entire course of the day, liquidity at the best ask price tended to vary a lot. Even in stocks with low volatility, it was observed that the standard deviation in liquidity throughout the day would often be well over one thousand shares of volume. What was also observed was that with a bit of noise, the best offer volume in very small windows had a standard deviation of less than one hundred shares. Therefore, we believe that an indication of the behavior of the dynamics of the best offer volume around time t will give the ANN helpful information in terms of making a prediction of the liquidity at time t.

Similar to $\mathcal{O}_t$ and $\mathcal{OD}_t$, we consider the variables $\mathcal{RM}_t$ and $\mathcal{RMD}_t$. These variables are defined as

$$\mathcal{RM}_t = \frac{\text{`MidPrice'}_t + \text{`MidPrice'}_{t-1} + \cdots + \text{`MidPrice'}_{t-4}}{5} \tag{4.19}$$

$$\mathcal{RMD}_t = \begin{cases} 1, & \text{`MidPrice'}_t - \text{`MidPrice'}_{t-1} < 0 \\ 0, & \text{`MidPrice'}_t - \text{`MidPrice'}_{t-1} \geq 0. \end{cases} \tag{4.20}$$

The average midprice over the past five ticks, and whether or not that midprice is decreasing. The variable 4.19 is far more useful for equities that have relatively low volatility. This is because for equities with lower volatilities, the algorithm can begin to recognize a relationship between low midprices similar to others that have been marked as trading ticks. We also

indicate in 4.20 as statistic so that the algorithm can learn that lower prices are more common when the mid-price is decreasing.

The variable $\mathcal{H}_t$ is a boolean variable indicating whether or not the Hawkes forecast of the midprice at time t as realized by the simulation of the process 4.4 using the Ogata thinning algorithm 4 with $\hat{\boldsymbol{\Theta}}_{\text{MLE}}$ is in the lower K-th percentile of midprices in the forecast. To best realize this, consider $\{X_t\}_{t=0}^{t=T}$ to be the sequence of the midprice of forecasts, and let $\mathcal{X}$ represent the times t^* such that X_{t^*} is in the lower $K-$th percentile of prices in the ordered sequence $\{X_t\}_{t=0}^{t=T}$. Now, we can define the variable $\mathcal{H}_t$ by

$$\mathcal{H}_t = \begin{cases} 1, & t \in \mathcal{X} \\ 0, & \text{otherwise.} \end{cases} \tag{4.21}$$

Finally, we include the realized volatility RV_t 4.1 into the ANN, or for the day we are finally using the trained model, the forecasted realized volatility as indicated by the Hawkes process given by 4.7. Note that this volatility will be constant among trading days, but since the ANN model will be trained on many months worth of data, the model will still be able to react to changes in volatility.

This neural network will help our agent accurately identify ideal trading times without using any past data. The next step is defining how much to trade when he arrives at an ideal trade tick. This is done by the reinforcement learning algorithm.

4.3 THE REINFORCEMENT LEARNING ALGORITHM

The reinforcement learning phase of this process marks the final step for the trader to obtain his ideal trading curve. In this step, we consider the actions of the trader in an optimal execution setting to be as they were originally introduced. Namely, the trader begins at time 0 with a goal of unwinding the portfolio position q_0 by time T. The entire focus of the problem is to find an optimal strategy v_t where $t \in [0, T]$ such that

$$\int_0^T v_t dt = -q_0.$$

We have considered in our models to approach the problem from the optimal purchase, instead of optimal liquidation, meaning that $q_0 < 0$ and we must therefore purchase q_0 shares by time T. The position at time t is given by q_t.

Our artificial neural network model from the previous section has, for our trader created a finite set $\{t_n^*\}_{0 \leq n \leq N}$ of trading times. From where the trader is standing at the current moment t^* in time, he does not know when the next trading time will happen. Therefore, N is not a known value at any given moment im time when the trader is making trades but is sure to be finite (there are T seconds in the interval $[0, T]$). This discretizes the structure of the trading curve for our model to be

$$\sum_{t_n^*} v_{t_n^*}.$$

Reinforcement learning has been used for optimal execution problems in finance in [27] and [28]. In [27], the authors tried to use reinforcement learning to locate the optimal execution times by including spreads and volumes into the state space. This is a good idea, but it greatly increases the number of states, as well as the state reward transition matrix's size and therefore computational expenditure to calculate it. Here, we do not need to include when to trade in our state space, or any elements of the state space that would cause a trader to make trades since our optimal trade times have already been captured by the ANN model. In [28], the actions taken by the trader comprise moving entire blocks of limit orders to a new price point over a very short time span. Our goal here with reinforcement learning is to decide how much to trade at specific, neural network recognized trading ticks with market

orders. We build our own reinforcement algorithm below, keeping in sync with the general theme of the few but present items in the literature, that the most important state aspects of reinforcement learning for optimal execution are that the time remaining to trade, and amount of inventory left to trade.

We recall that in a reinforcement learning task, the agent, in our case the trader finds himself in a state $s_{t_n^*}$. While in this state he performs an action $A \in \mathcal{A}(s_{t_n^*})$, which lands him in a new state $s(t_{n+1}^*)$ at time t_{n+1}^* and gives the trader a reward r_{n+1}.

We begin the description of this model for optimal execution by first defining the state space $\mathcal{S}$ for the model. This requires two parameters, namely T, and q_0.

We maintain a similar structure to the reinforcement learning algorithm as was done in the initial data driven approach.

$$\mathcal{S} = \{\mathbb{X}, \mathbf{T}\} \tag{4.22}$$

$$\mathbb{X} = \{X_1, X_2\} \tag{4.23}$$

$$\mathbf{T} = \{t_1, t_2, t_3\} \tag{4.24}$$

- t_1: Represents the time interval $\left[0, \frac{1}{3}T\right)$
- t_2: Represents the time interval $\left[\frac{1}{3}T, \frac{2}{3}T\right)$
- t_3: Represents the time interval $\left[\frac{2}{3}T, T\right)$
- X_1: Amount remaining to be purchased $|q_t| \in \left(\frac{1}{2}|q_0|, |q_0|\right]$
- X_2: Amount remaining to be purchased $|q_t| \in \left[0, \frac{1}{2}|q_0|\right)$

We now fully define $\mathcal{S}$, a set of cardinality 7. Recall, that this cardinality plays majorly into the computation of our ideal value and policy function, and determines the size of our state/action to state/reward transition matrix.

$$\mathcal{S} = \{(X_1, t_1), (X_1, t_2), (X_1, t_3), (X_2, t_1), (X_2, t_2), (X_2, t_3), \} \tag{4.25}$$

$$\mathcal{S}^+ = \{(X_1, t_1), (X_1, t_2), (X_1, t_3), (X_2, t_1), (X_2, t_2), (X_2, t_3), \text{TERMINAL}\} \tag{4.26}$$

The TERMINAL state occurs when the trader has reached $q_t = 0$, and completed the total purchase amount q_0.

In previous builds of the model we allowed the trader's action set to include an option to not trade. In the rendition we present for this dissertation the agent is no longer allowed to choose not to trade at the ANN discovered trade times t^*. The reasoning for this is that the criteria for locating trading times are strong and the ANN model tells the trader exactly when conditions are good to trade. We weigh whatever poor outcomes could occur as a result of the trader being forced to trade and making poor trades here and there against the penalty he incurs if he doesn't make the full purchase by time T. In the sense of the optimal execution framework, this means that the trader has to make a large market order at time $T - \varepsilon$ for the remainder of what he wishes to trade. This may well walk the book to a large degree if the $|q_{T-\varepsilon}|$ is large.

What is not often intuitive about machine learning is that the algorithms will learn the best strategy only in the context of their learning environment. In this case the learning is the state space and reward cycle. In a market with high intraday volatility and activity, the algorithm may notice that it is constantly receiving the negative reward for walking the limit order book when the agent chooses a trading action. It is therefore not at all uncommon for the algorithm to learn that not making trades is the best course of action in a majority of the states. This would in turn cause the agent to do nothing at the given trading times t^* and place a massive market order at the very end of his trading cycle $T - \varepsilon$ to incur a large loss. We thus define the agent's action set to be

$$\text{for } t^* \in \{t_n^*\}_{0 \leq n \leq N},\ \mathcal{A}(s_{t^*}) = \{\text{Buy Low, Buy High}\}\,. \tag{4.27}$$

Where these actions are defined as

1. Buy Low: $\mathcal{L}_{\text{low}} < v_{t^*} < \mathcal{L}_{\text{high}}$
2. Buy High: $v_{t^*} > \mathcal{L}_{\text{high}}$.

This parameter $\mathcal{L}_{\text{high}}$ serves as an upper bound in the algorithm for the threshold of making a "high" trade. The parameter $\mathcal{L}_{\text{low}}$ serves as a technical bound because without it, the trader could essentially choose to not make a trade. Higher values of $\mathcal{L}_{\text{low}}$ will force the agent into more aggressive trading curves. Lower values (more suited to larger values of T) will allow the trader to be more conservative in trading.

We define the reward set for the trader as a finite set containing three basic rewards

$$\mathcal{R} = \{r_{\text{match}}, r_{\text{slip}}, r_{\text{shock}}\}. \tag{4.28}$$

All values $r \in \mathcal{R}$ take values in $\mathbb{Z}$ and are given to the trader based on the following criteria:

- r_{match} is given at time t^*_{n+1} if the action Buy Low was taken in time t^*_n and the trade was executed completely at the best price offer price available to the trader.
- r_{slip} is given at time t^*_{n+1} if the trade resulting from the action Buy High or Buy Low at time t^*_n walked the limit order book.
- r_{shock} is given at time t^*_{n+1} if the action Buy High was taken in time t^*_n and the trade was executed completely at the best price offer price available to the trader.

An example of such a reward set is

$$\mathcal{R} = \{r_{\text{match}} = +3,\ r_{\text{slip}} = -1,\ r_{\text{shock}} = -10\}\,. \tag{4.29}$$

The reward set is an input parameter in the algorithm, but there are some stipulations on the input of the parameters. Namely

$$r_{\text{slip}} < 0, \qquad r_{\text{match}}, r_{\text{shock}} > 0, \qquad r_{\text{match}} < r_{\text{shock}}. \tag{4.30}$$

We restrict $r_{\text{slip}} < 0$ so that the algorithm punishes the trader for making trades that the book. r_{match} and r_{shock} are positive rewards because the trader has executed his trade at the best price avaialable to him in the limit order book at the time. We also wish to make $r_{\text{match}} < r_{\text{shock}}$ to ensure that the reward that the trader gets for executing a larger trade at the best price is more than the reward for a smaller trade at the best price.

All in all, this entire model has three rewards, two actions, six non-terminal states, and seven total states. Therefore, the adequate state/action - state/reward transition matrix is a 12 by 21 matrix meaning we must calculate 252 individual transition probabilities:

$$p\left(\{(X_j, t_j), r_j\} | \{(X_i, t_i), r_i\}\right). \tag{4.31}$$

4.3.1 Using Trading Data to calculate the State/Action - State/Reward Transition Matrix

The calculation of the transition matrix depends on the availability of good data. If data is extremely lacking in detail, one does have the ability to make educated guesses as to the transition probability values. For instance, since the agent only has options to buy, then it is not possible to transition from an X_2 state to an X_1 state, and since time increases from 0 to T, it is not possible to transition from a t_2 state to a t_1 state. Techniques such as this are possible when all else fails, but not really feasible to get a good read on how an actual equity behaves.

As a result of doing this research, we have access to the TAQ database which records trades and quotes that take place throughout the market. From this database, we have devised a simple algorithm to generate the state/action to state/reward transition matrix $\mathbf{A}$ where $\mathbf{A}_{ij}$ represents the probability of transition from state/action pair i to state/reward pair j.

As mentioned earlier, this is the final stage in the process of building the trading curve. Therefore, we will be using all of the data that has been used to train the ANN model to train the reinforcement learning model. We begin the process by considering an arbitrary state/action pair

$$SA_i = \{(X_i, t_i), a_i\} \tag{4.32}$$

where $(X_i, t_i) \in \mathcal{S}$ as defined by 4.25 and $a_i \in \mathcal{A}$ from 4.27. From the TAQ data that has been used to train the ANN model, which we will call TQ_{ANN}, we select a subset $TQ^i_{\text{ANN}} \subset TQ_{\text{ANN}}$ where TQ^i_{ANN} represents all of the trades that take place at ideal trading times t^* in the TAQ data that is representative of the state/action pair SA_i. For instance, if

$$SA_i = \{(X_1, t_1), \text{Buy Low}\},$$

then TQ^i_{ANN} would consist on data for every trade from TQ_{ANN} such that the time stamp was in the range for t_1 and the trade size was in the interval $[\mathcal{L}_{\text{low}}, \mathcal{L}_{\text{high}})$. This creates the set TQ^i_{ANN} such that each trade $\mathcal{T} \in TQ^i_{\text{ANN}}$, there is an associated array of values

$$\mathcal{T}_{\text{values}} = \left[\mathcal{T}_{\text{size}}, \mathcal{T}_{\text{time}}, \mathcal{T}_{\text{price}}, Q^{BO}_{\mathcal{T}}, X^{\mathcal{T}}_{\mathcal{U}[0,1]}, \mathcal{T}_{t^*}\right]. \tag{4.33}$$

For these values, $\mathcal{T}_{\text{size}}$ represents the size of the trade that was made (shares), $\mathcal{T}_{\text{time}}$ represents the time $[0, T]$ that the trade was made, and $\mathcal{T}_{\text{price}}$ is the price (per share) that was paid in the trade. $Q_{\mathcal{T}}^{BO}$ represents the best offer price in the limit order book for at time $t = \mathcal{T}_{\text{time}}$.

Knowing that TQ^i_{ANN} has been marked via training for the ANN model, let $\mathcal{T}_{t^*_{\text{day}}}$ represent the largest time t^* in the day that T takes place in the period t_i. By day, what we mean is the cycle of time $[0, T]$ of training data that T is represented in that has been broken up into the periods of t_1, t_2, t_3. We will train the model on many months and possibly even years worth of data. This means that there will be several intervals of time $[0, T]$ that the model is trained on. Each one of these intervals will have optimal trading times t^* identified. Therefore, since every trade $\mathcal{T} \in TQ^i_{\text{ANN}}$ exists in one of these 'days' of intervals $[0, T]$, there is a final optimal trading time in the period t_i of that day. We call this $\mathcal{T}_{t^*_{\text{day}}}$. We want to indicate whether or not $\mathcal{T}$ is this last trading time of the day and to do so we define the boolean variable $\mathcal{T}_{t^*}$ by

$$\mathcal{T}_{t^*} = \begin{cases} 1 & T_{\text{time}} < \mathcal{T}_{t^*_{\text{day}}} \\ 0 & T_{\text{time}} = \mathcal{T}_{t^*_{\text{day}}} \end{cases}. \tag{4.34}$$

The value $X^T_{\mathcal{U}[0,1]}$ is unique in the sense that it is not readily extracted from the data that is brought in from the ANN model training data. This value is actually assigned at this point in the process. In order to turn the concept of the optimal execution problem into one suitable for reinforcement learning we had to create a Markov decision process (MDP) with finite states. This has restricted us to treating the amount of money that remains to be traded $|q_0|$ into a two state space; $\{X_1, X_2\}$. In order to calculate the transition probabilities, we want to simulate a position within each state X_i that the trader needs to trade in order to get out of the state X_i and into either X_{i+1} or TERMINAL. In order to do this, we assign to each trade T a simulated position within the state X_i. We do this by simulating

$$u_T \sim \mathcal{U}_{[0,1]} \tag{4.35}$$

$$X^T_{\mathcal{U}[0,1]} = \frac{|q_0|}{2} * u_T. \tag{4.36}$$

and we allow $X^T_{\mathcal{U}[0,1]}$ to represent the amount remaining to be traded to transition purchase states from X_i to X_{i+1} (where $X_3 = \text{TERMINAL}$) before the trade T takes place.

The reason for the consideration of simulating this random variable is that the $X-$states can theoretically occur at any time and at any trade in the trading data. The trader could reach the TERMINAL purchase state in time interval t_1 if he trades a lot at the beginning. Therefore, we randomly assign this purchase position and will simulate whether trade T takes it to the next step or not.

We are now in a position to calculate the prediction $\tilde{\mathcal{T}}_{SR}$ for the state reward pair that taking action a_i at trade $\mathcal{T}$ in state (X_i, t_i).

$$\tilde{\mathcal{T}}_{SR}(\mathcal{T}) = \begin{cases} \{(X_i, t_i), r_{\text{slip}}\}, & \mathcal{T}_{\text{price}} > Q_{\mathcal{T}}^{BO} \text{ and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} > \mathcal{T}_{\text{size}} \text{ and } \mathcal{T}_{t^*} = 1 \\ \{(X_i, t_i), r_{\text{match}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{Buy Low and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} > \mathcal{T}_{\text{size}} \\ & \text{and } \mathcal{T}_{t^*} = 1 \\ \{(X_i, t_i), r_{\text{shock}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{Buy High and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} > \mathcal{T}_{\text{size}} \\ & \text{and } \mathcal{T}_{t^*} = 1 \\ \{(X_{i+1}, t_i), r_{\text{slip}}\}, & \mathcal{T}_{\text{price}} > Q_{\mathcal{T}}^{BO} \text{ and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} < \mathcal{T}_{\text{size}} \text{ and } \mathcal{T}_{t^*} = 1 \\ \{(X_{i+1}, t_i), r_{\text{match}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{Buy Low and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} < \mathcal{T}_{\text{size}} \\ & \text{and } \mathcal{T}_{t^*} = 1 \\ \{(X_{i+1}, t_i), r_{\text{shock}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{Buy High and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} < \mathcal{T}_{\text{size}} \\ & \text{and } \mathcal{T}_{t^*} = 1 \\ \text{Additionally if } t_i = t_1, t_2: & \\ \{(X_i, t_{i+1}), r_{\text{slip}}\}, & \mathcal{T}_{\text{price}} > Q_{\mathcal{T}}^{BO} \text{ and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} > \mathcal{T}_{\text{size}} \text{ and } \mathcal{T}_{t^*} = 0 \\ \{(X_i, t_{i+1}), r_{\text{match}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{Buy Low and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} > \mathcal{T}_{\text{size}} \\ & \text{and } \mathcal{T}_{t^*} = 0 \\ \{(X_i, t_{i+1}), r_{\text{shock}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{Buy High and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} > \mathcal{T}_{\text{size}} \\ & \text{and } \mathcal{T}_{t^*} = 0 \\ \{(X_{i+1}, t_{i+1}), r_{\text{slip}}\}, & \mathcal{T}_{\text{price}} > Q_{\mathcal{T}}^{BO} \text{ and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} < \mathcal{T}_{\text{size}} \text{ and } \mathcal{T}_{t^*} = 0 \\ \{(X_{i+1}, t_{i+1}), r_{\text{match}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{Buy Low and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} < \mathcal{T}_{\text{size}} \\ & \text{and } \mathcal{T}_{t^*} = 0 \\ \{(X_{i+1}, t_{i+1}), r_{\text{shock}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{Buy High and } X_{\mathcal{U}[0,1]}^{\mathcal{T}} < \mathcal{T}_{\text{size}} \\ & \text{and } \mathcal{T}_{t^*} = 0 \\ \text{Additionally if } X_i = X_1: & \\ \{\text{TERM}, r_{\text{slip}}\}, & \mathcal{T}_{\text{price}} > Q_{\mathcal{T}}^{BO} \text{ and } \left(\frac{|q_0|}{2} + X_{\mathcal{U}[0,1]}^{\mathcal{T}}\right) < \mathcal{T}_{\text{size}} \\ \{\text{TERM}, r_{\text{match}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{BL and } \left(\frac{|q_0|}{2} + X_{\mathcal{U}[0,1]}^{\mathcal{T}}\right) < \mathcal{T}_{\text{size}} \\ \{(\text{TERM}, r_{\text{shock}}\}, & \mathcal{T}_{\text{price}} \leq Q_{\mathcal{T}}^{BO} \text{ and } a_i = \text{BH and } \left(\frac{|q_0|}{2} + X_{\mathcal{U}[0,1]}^{\mathcal{T}}\right) < \mathcal{T}_{\text{size}} \end{cases} . \tag{4.37}$$

We will perform the test for classification given by 4.37 for all trades $\mathcal{T} \in TQ^i_{\text{ANN}}$. Let $|TQ^i_{\text{ANN}}|$ represent the cardinality of the set TQ^i_{ANN}, which tells us exactly how many trades we have classified. For every state/reward pair $\{s, r\}$ let

$$\{s, r\}_{SA_i} = \sum_{\mathcal{T} \in TQ^i_{\text{ANN}}} \mathbb{1}_{\tilde{\mathcal{T}}_{SR}(\mathcal{T}) = \{s,r\}} \tag{4.38}$$

In other words, 4.38 represents all of the trades in SA_i that are predicted by 4.37 to be in the given state reward pair $\{s, r\}$. We now have the element of the transition matrix $\mathbf{A}$ to be

$$p\left(s, r | \{(X_i, t_i), a_i\}\right) = \frac{\{s, r\}_{SA_i}}{|TQ^i_{\text{ANN}}|}. \tag{4.39}$$

4.3.2 Forward to the Optimal Trading Curve

Now that the state/action-state/reward transition matrix is defined all that we need to do in order to obtain the optimal policy is to first put forward a starting policy, reward set, high/low trading thresholds, and $|q_0|$ for instance:

$$\begin{aligned}
\pi_{\text{initial}} = \big\{&(X_1, t_1): \text{ Buy Low }, \\
&(X_1, t_2): \text{ Buy Low }, \\
&(X_1, t_3): \text{ Buy Low }, \\
&(X_2, t_1): \text{ Buy Low }, \\
&(X_2, t_2): \text{ Buy Low }, \\
&(X_2, t_3): \text{ Buy Low }, \big\} \\
\mathcal{R} = &\{r_{\text{match}} = +3,\ r_{\text{slip}} = -1,\ r_{\text{shock}} = -10\}. \\
\mathcal{L}_{\text{high}} = &700 \\
\mathcal{L}_{\text{low}} = &100. \\
|q_0| = &5000
\end{aligned}$$

Then building matrix $\mathbf{A}$ as defined by 4.37, we may perform 3 and obtain the optimal policy π^*. Since the trader's state space is mutually exclusive and exhaustive for all situations he finds himself in throughout executing the purchase, he will perform the relevant action as indicated by π^* in the state that matches his optimal trading time t^* and amount remaining to be purchased $|q_t|$.

4.4 THE COMPLETE ALGORITHM

Here we present the complete algorithm for the trader to unwind a portfolio position of $-q_0$ in a given equity over a time interval $[0, T]$.

We consider the interval we wish to train in as 'Day', and all of the previous time intervals that we are using to train our model as 'TrainDays.'

1)**Data**: Input T, $|q_0|$, TrainDays, Day, K (percetile), TAQ Data

2)Tag ideal trading times in TAQ data: ;

for *day in TrainDays* **do**

  for *time in day* **do**

    if *Midprice is in the bottom K-th percintile in day and offer volume is above median in day:* **then**

      time_{t^*} = True

    end

    else

      time_{t^*} = False

    end

  end

end

3)Simulate Hawkes Process for each day;

for *day in TrainDays* **do**

  Use the ideal trading times marked t^* to find $\hat{\mathbf{\Theta}}_{\text{MLE}}^{\text{Day}}$ using 4.16;

  Simulate the Hawkes Process Price Simulation X_t^{Day} defined by $\hat{\mathbf{\Theta}}_{\text{MLE}}^{\text{Day}}$ using 4.1.1

end

4) Train The Neural Network Model ;

for *day in TrainDays* **do**

  Use X_t^{Day} to train an Artificial Neural Network (sequential, 5 layers, 100 nodes each to output, ReLu/Softmax-input/output Tensorflow Backend) as shown in 4.2 ;

  Save the final model as TrainedANN

end

5)Perform The Reinforcement Learning Algorithm;

for *day in TrainDays* **do**

  Use 3 with the framework of 4.37 to produce the optimal policy π^*

end

6)Trade

Algorithm 5: Complete Optimal Execution for Market Orders

5.0 THE LIMIT ORDER MODEL

Up to this point, our algorithms have devised strategies for the trader to post market orders at various optimal times to make them. We are now going to introduce the concept of the trader placing a new type of order known as limit orders in the market. The optimal execution framework however, stays the same. We recall that in the optimal execution problem the trader is faced with the problem of purchasing or selling a given (large) amount of shares q_0 of a certain stock $\mathbb{X}$ by time T. Given the way that our limit order book market model is set up, by doing such a task with market orders, the trader will always be buying his stock from the offer side of the limit order book or selling from the ask side. Thus, no matter how low the offer price goes, he is always making his purchases from the side of the spread that is not to his advantage in the sense that at any given time, the other side of the limit order book is showing better prices.

As we have explained before, when a market order is placed, it is routed through the limit order book to find the best matching limit order. A market order placed to buy must find the best limit order that has been placed by another player in the market, and execute. Market orders to sell always match with limit orders placed on the bid side. In an arbitrage free model, the best bid price is always lower than the mid-price and the best ask price. Thinking in this manner, we know that if our goal is to sell a stock by time T, that there are people in the market selling this stock for more money that us if we place a market order for it, namely those who are placing limit orders to sell the stock that are executing whenever an adequate market buy order enters the market.

The model we introduce here seeks to optimze the utility of the trader by placing adequate limit orders in the market that he hopes will execute by time T. When he posts a limit order in the market he specifies to the exchange a price point, and a size. He is given a position

in the limit order book according to the principal of FIFO (first in first out). Thus, if other traders have posted limit orders in the limit order book at the same price level as our trader, then our order will not execute until all of the other orders ahead of us have been executed. In order to explore the trader's interactions with limit order placement, we begin with the model found in chapter 7 of [3].

5.1 THE TRADITIONAL LIMIT ORDER MODEL

In the traditional setup of the limit order book model as found in [3], the trader has to execute the sale of q_0 shares of a certain equity by time T. The mid-price of the equity follows the standard SDE model of

$$S_t = S_0 + \sigma dW_t \tag{5.1}$$

where W_t is a standard Brownian motion. In this model, the limit orders are all of a standard size (can be assumed to be of size 1), and the trader's control process is where he places the limit order at a given time t. Thus, the control process of the trader is denoted by the variable $\delta_t = (\delta)_{0 \leq t \leq T}$. This depth is a depth based on the mid-price. Therefore, the limit order is placed at the price point $S_t + \delta_t$.

The model simulates the expected execution of the order by considering two processes M_t and N_t^δ.

- M_t denotes a Poisson process that corresponds to the number of market buy orders from other traders that have arrived by time t.
- N_t^δ denotes a counting process of the arrival of the number of market buy orders which lift the order we have placed at $S_t + \delta_t$. We note that this process is not a Poisson process itself.

The limit order executes with expected probability $P(\delta) = e^{-K\delta}$. In order to make an optimization problem for the trader that will result in a Hamilton-Jacobi-Bellman PDE we

consider the cash process of the trader who has placed a limit order at depth δ at time t to be

$$\begin{aligned} X^{\delta} &= (X_t^{\delta})_{0<t<T} \\ dX_t^{\delta} &= (S_t + \delta_t) dN_t^{\delta} \\ Q_t^{\delta} &= \mathcal{N} - N_t^{\delta}. \end{aligned}$$

Here Q_t^{δ} represents the agent's inventory to be liquidated. This yields the agent's optimization problem of

$$\begin{aligned} H(x,S) &= \sup_{\delta \in \mathcal{A}} \left[X_{\tau}^{\delta} + Q_{\tau}^{\delta}(S_{\tau} - \alpha Q_{\tau}^{\delta}) |\ X_{0^-}^{\delta} = x, S_0 = S, Q_{0^-}^{\delta} = \mathcal{N} \right] \\ \tau &= \min(T, \min(t : Q_t^{\delta} = 0)) \\ H(t,x,S,q) &= \sup_{\delta \in \mathcal{A}} \mathbb{E}_{t,x,S,q} \left[X_{\tau}^{\delta} + Q_{\tau}^{\delta}(S_{\tau} - \alpha Q_{\tau}^{\delta} \right]. \end{aligned}$$

We take a markedly different approach to this problem. One that begins with a statistical approach to the probability of execution function.

5.2 A LOGISTIC REGRESSION MODEL FOR THE PROBABILITY OF EXECUTION FUNCTION

One of the aspects of the traditional model that we would like to improve upon is the probability function for execution of a limit order at a certain depth. To build the framework, consider a binary random variable X_t such that

$$X_t(\omega) = \begin{cases} 1 & \text{if the limit order placed at time } t \text{ executes.} \\ 0 & \text{if the limit order placed at time } t \text{ fails to execute.} \end{cases} \tag{5.2}$$

In the traditional limit order model, the probability that a limit order executes given that it is placed at a position δ from the spread is

$$P(X_t(\omega) = 1 | \delta) = \exp(-K\delta) \tag{5.3}$$

for some parameter K. This model of probability allows for convenience in building an adequate HJB model, and is very sensible in that the probability that a limit order executes will decrease quickly as delta is increased since it will be placed deeper into the limit order book. Every time the order is placed at a deeper price point, all of the limit order volume preceding it must be taken out by market orders. However, once one starts to switch equities for the optimal liquidation, so many market characteristics will be levied into the correct choice of parameter K. Since our approach does not include an HBJ equation, we will free ourselves from this structure of the model.

Since the traditional model as given in [3] considers the optimal liquidation problem, so too will we. This is a departure from our work in the market order models when the goal of the agent was to optimally purchase. Our probability function will take inputs consisting not only of depth. This is becuase, at any current time, we can observe more characteristics than just the price spread. Our probability function takes the form of

$$P(X_t(\omega) = 1|\delta, \mathcal{V}, \sigma, \eta, \tau) \tag{5.4}$$

given by a a logistic regression model. The volatility, σ: As we have seen throughout this research, volatilities can be measured in different ways such as historical, GARCH, implied, and realized volatility. We again employ realized volatility here. Volatility is a measure of the stock price's propensity to change. Thus, given a blue chip stock like Johnson and Johnson, considered in juxtaposition with a cryptocurrency stock like Bitcoin, we will see a massive difference in how much the price will change in a given time period. Thus, the chance that a limit order placed ten dollars below the spread, has, for Johnson and Johnson virtually zero chance of execution by the end of the day. However, with Bitcoin, a stock that in recent history has jumped and fallen thousands of dollars in a given trading day, the chance of a limit buy order executing ten dollars below the spread is significantly higher. We thus consider the input of an intraday volatility measure to be invaluable condition in the computation of the probability of limit order execution.

We also condition on the liquidity, η. This statistic is important in that when we place a limit order at a given price level, we will be put into the last place in the queue of limit orders. Thus, in a stock with less liquidity (holding order flow to be constant) measured in

the sense of smaller volume of limit orders at each tick, a limit order will execute with higher probability.

Depth, δ is considered as is the case in the original model. The depth at which a limit order is placed is vital to its chances of being executed. A limit sell order placed ε below the best ask price will theoretically execute instantly upon the next market sell order's arrival. A limit sell order placed ten dollars above the ten year high of the stock has almost no chance to execute other than the event of a shock. An example would be a biotech stock for a company that has just announced that they have found the cure for cancer. In less extreme situations, the depth is still a vital input to condition on in that every cent further the limit order is placed above the best ask price, an entire new stack of limit order volume must be wiped out in order for the limit order to execute.

The size $\mathcal{V}$ of our limit plays a significant role here as well. A limit order placed with large volume $\mathcal{V}$ requires more opposing market orders to come in, in order to make it fully execute.

Time left to trade $\tau = T - t$ is a two faceted variable to condition on from our perspective. In our problem specifically, we have to deal with the problem of the time horizon running out. Thus, a limit order placed with less time left in the time horizon runs a greater risk of not executing for the simple reason that there is not enough time to allow for price fluctuations in the market that would cause the order to execute. On the other hand, as we have talked about with respect to the limit order model, it is to the advantage of the practitioner in these machine learning algorithms to traing the algorithm on intervals that possess a certain sense of regularity. Thus, if we choose our time intervals to train the algorithm on in the form of $[0, T]$ where 0 and T represent the opening and close of the trading day respectively, our algorithm will acquire some degree of the measure of average order flow for various times throughout the day. Without a bit of order flow, the limit orders have little to no chance of execution. This will be learned by the algorithm via training data which, as we will see is tagged to show which orders executed and which did not.

5.3 STRUCTURE OF THE ALGORITHM

Our goal is to generate the probability function 5.4 via the machine learning technique of logistic regression. To reiterate from the background section, despite logistic regression having the word "regression" in the title it is in actuality a method for classification tasks. What it does by regression is the formation of a decision boundary in order to choose which to choose classifiers if they are above or below the boundary. We will perform such a regression across four variables, namely the ones described above: δ, η, σ, and τ.

5.3.1 Building the Training Data

At each time t, where for our purposes t is measured in seconds, we observe the following four statistics.

σ_t is realized as the realized volatility as defined by 4.1 at the current trading day. Therefore, σ_t is constant throughout each trading day. In order to train the model, we will use the realized volatility as it is calculated via 4.1. This is because when building the training data, we can look both backwards, and forwards in time.

η_t is interpreted as the 'OFRSIZ' statistic at the current time series tick. We note that this is the volume that is present at the best ask price in the limit order book at time t. An ideal algorithm would include the volume present at the depth where the limit order is placed. Unfortunately, we do not have such data at our disposal, so we will have to make do with the 'OFRSIZ' statistic.

τ_t is realized as the current time tick's difference with the final tick of the day. This one day deadline is chosen to make our job computing it a bit easier. We note that the extension to longer periods of time for the training of the algorithm is not difficult, just a bit more computationally involved when one considers time series computations.

In order to calculate, the depth δ_t, the user of the algorithm would be wise to do some exploratory data analysis first in order to appropriately set the variable $\hat{\delta_{\max}}$, such that $\hat{\delta_{\max}} > 0$, which represents the maximum depth limit order that the trader would consider.

Following this, δ_t is given as

$$u_t \sim \mathcal{U}_{[0,1]} \tag{5.5}$$

$$\delta_t = \hat{\delta_{\max}}. \tag{5.6}$$

If the trader sets a $\hat{\delta_{\max}}$ value too high, then he is wasting quite a bit of of training data, as a lot of these limit orders will never execute, and the ones that do will be placed into a smaller window of viable 5.9. Similarly, choosing $\hat{\delta_{\max}}$ too low would cause situations in the training data in which almost every limit order placed converges, leading the algorithm to a situation in which it possesses quite a bit of class imbalance. We recommend choosing a value of $\hat{\delta_{\max}}$ of

$$\hat{\delta_{\max}} = \max_{t\in[0,T]} S_t - \text{median}_{t\in[0,T]}(S_t). \tag{5.7}$$

The variable $\mathcal{V}_t$ is also randombly selected, like depth via a uniformly distributed random variable by

$$v_t \sim \mathcal{U}_{[0,1]} \tag{5.8}$$

$$\mathcal{V}_t = v * \hat{\mathcal{V}_{\max}}. \tag{5.9}$$

We recommend choosing $\hat{\mathcal{V}_{\max}}$ to be the 75th percentile trade size from the TAQ data over the interval. This will provide the simulated agent a degree of a familiarity with the market from the data, and not make outrageously large trades.

In order to produce a logistic regression, we have to be able to classify whether or not the limit order executed. So, in order to complete our training data we must define $X_t(\omega)$ for each time t. This is an issue that is actually the most computationally expensive portion of the process. We need to decide based on our trade data how we will realize the execution of a limit order. We make the following observations.

We are dealing with second sampled time series data. This means that when we place a limit order at that second based on the current spread at that second, that we will be in the back of the queue of that price point. Therefore, as we look forward in time, we assume that our limit order placed at point $S_t + \delta_t$ will execute if aggregated trades made at price

points strictly lower than the position of our limit order after our limit order is placed and before the end of the trading day exceed the volume of our limit order.

The question of whether or not we could have our limit order execute at the given depth of the limit order is a legitimate one. We decide that in our case, this is not possible because the data that we are working with is a large, dense record of trades and quotes. Thus, our limit order is not actually present in the market that it reflects. Therefore, we have to consider the first trade executed at a lower level to be the first market order that could have lifted our limit order and caused the desired trade to execute. Therefore, we search for volume at lower values than our limit order. Let $\mathcal{T}_{(t,T]}$ represent the volume of shares traded in the market in the interval $(t, T]$. We can now define

$$X_t(\omega) = \begin{cases} 1 & \mathcal{T}_{(t,T]}\mathbb{1}_{\text{price}>S_t+\delta_t} \geq \mathcal{V}_t \\ 0 & \text{otherwise} \end{cases}. \tag{5.10}$$

We can now define the training data for the logistic regression algorithm for all times t in our training set. Performing the logistic regression algorithm defined in the background chapter on the trading set, we now obtain the definition of 5.4.

5.4 COMPLETING THE OPTIMAL EXECUTION PROBLEM FOR LIMIT ORDERS

Now that we have the training data set, and a trained function for 5.4, we turn our sights toward the actual goal of the optimal execution problem, which is to find the optimal trading curve. In this consideration of the optimal execution problem, our trader is faced with the problem of unwinding the position q_0 such that $q_0 > 0$ in the case of optimal liquidation. His only actions in this model are the ability to place limit orders at every time t at depth $S_t + \delta_t$ at size $\mathcal{V}_t$. His action, at every time is to select the controls that maximize the utility function

$$U(\delta_t, \mathcal{V}_t) = (S_t + \delta_t)\mathcal{V}_t p\left(X_t(\omega) = 1 | \delta_t, \mathcal{V}_t, \sigma_t, \eta_t, \tau_t\right) + \frac{q_t}{q_0}(\mathcal{V}_t) \tag{5.11}$$

In 5.11, what we are asking the trader to do is to maximize the sale he would make with the limit order at depth δ_t of size $\mathcal{V}_t$ while weighing the *penalty* as expressed by

$$\frac{q_t}{q_0}(\mathcal{V}_t). \tag{5.12}$$

The penalty is something unique to the limit order model in that it represents the *penalty* that would incur as a result of *none* of the limit orders the trader places executing, making the trader have to place a market order to liquidate q_t at time $T - \varepsilon$. α is what we refer to as an *impact parameter*, and it represents how much a large order has the propensity to walk the book. Naturally, the strategy taken of $(\delta_t^*, \mathcal{V}_t^*)$ such that

$$(\delta_t^*, \mathcal{V}_t^*) = \text{argmax}_{(\delta_t, \mathcal{V}_t)} U(\delta_t, \mathcal{V}_t)$$

will make the trader more agressive when there is more to trade (q_t closer to q_0), and more conservative when q_t is closer to zero. Maximization of this utility function defines the trader's action at each time. We are ready to introduce the optimal liquidation algorithm.

1)**Data**: Input Training TAQ Data, $\hat{\delta_{\max}}$, $\hat{\mathcal{V}_{\max}}$

, q_0 ;

2)Build the training data for the logistic regression function using 5.10. Use this to define 5.4, p^* ;

2)Use p^* to place limit orders ;

for t *in* $[0, T]$ **do**

place the limit order at ;

$(\delta_t^*, \mathcal{V}_t^*) = \text{argmax}_{(\delta_t, \mathcal{V}_t)} U(\delta_t, \mathcal{V}_t)$;

where U is defined by p^*. ;

if *in* $(0, t)$, *any of the previous limit orders have executed* **then**

$q_t = q_0 -$ Volume Executed ;

end

if $q_t = 0$ **then**

cease limit order placement

end

end

Algorithm 6: Optimal Limit Order Placement Algorithm

A note on the variables for $p^* = p^* \left(X_t(\omega) = 1 | \delta_t, \mathcal{V}_t, \sigma_t, \eta_t, \tau_t\right)$ in the algorithm above. Both η and τ are readily observable. However, with regards to σ, it has been directly calculated for all of the training data. This is not possible for the current problem that is happening in real time. Therefore, it is computed via 4.7 given 4.16 from the training data.

6.0 COMPUTATIONAL SIMULATIONS

6.1 SIMULATIONS FOR THE HAWKES APPROACH

We will walk through the entire algorithm leading up to the optimal trading curve for the Hawkes model in detail. We recall that at the beginning, we have the goal of the trader, that is to unwind a position q_0 by time T in an optimal fashion. In order to achieve this, we employ the algorithm given in 5.

6.1.1 Building the TAQ based LOB data

The Trade and Quote (TAQ) data that we have available to us via the WRDS database given by [29] is separated initially into two files; trades and quotes. The trades file contains information on all trades made in the market at time t. This data is available to us as it appears in 6.1. We see that the data, listed above for the trades in equity 'MCD' appears discretized by the second. Listed are the price and size of the trade. This price is the average price paid per share. In market microstructure theory, we know that depending on things like orders walking the book, the agent may pay different prices for different shares in each order. The price statistic given is the average price in terms of

$$\text{PRICE} = \frac{\text{Amount Paid (\$)}}{\text{Number of Shares}}.$$

The condition of the trade is listed in the TAQ user data in order to describe what type of condition the trade occurred under. We are concerned with the microstructure of the limit order book in a much more general sense than the detail that this statistic provides. We therefore disregard it in our studies. The column 'EX' denotes the exchange that this trade

SYMBOL	DATE	TIME	PRICE	SIZE	COND	EX
MCD	20110103	7:32:26	71.28	19	T	N
MCD	20110103	7:39:03	72.50	10	S	N
MCD	20110103	7:45:23	73.23	400	K	P
MCD	20110103	7:45:23	74.63	150	L	P
MCD	20110103	7:46:35	75.13	700	F	N

Table 6.1: This table represents trades for the stock 'MCD' on January 3rd, 2011. Due to preservation of data integrity, the true values for these times have been changed from their true values, and this figure is given only with the intent to show the reader the structure of the data.

took place on. Despite trade re-routing being the standard practice as described in [3], we find it best to isolate our studies to those in a single exchange so that we can best monitor the price movements. Therefore we restrict our data to only the NYSE exchange indicated as EX = 'N' and consider only times in which the NYSE is open; namely 9:30:00 a.m. until 4:30:00 p.m. EST. during non-holiday week days. Our data for quotes appears much in the same manner. We see in 6.2 that multiple quotes of the equity can exist for any given time. This is fine, but later we will need to isolate for the purposes of our studies the measure of the best fit quote for each second. We also are given the BID and ASK statistics. These are vital for our studies. We recall from a previous section that our goal is to build a model of the limit order book, and that this complete limit order book model would serve as the ideal data with which to train our model. This is not possible given the data but noting that a common model given in both [19] as well as [3] is a block shaped limit order book model at the best bid and ask price. We, at our disposal with the TAQ data are able to give a data justified replica of this block shaped model. In order to respect the limitations of our data we will place trading limitations on the trader to keep his activity around the best bid/ask price volume. The data for both bid size and ask size is given in 'round lots' or units of 100

SYMBOL	DATE	TIME	BID	OFR	BIDSIZE	OFRSIZE	MODE	EX
MCD	20110301	4:00:00	72.20	73.05	3	3	12	P
MCD	20110301	4:00:05	72.30	73.05	5	5	12	P
MCD	20110301	4:00:05	72.90	73.05	7	7	12	P
MCD	20110301	4:00:05	72.95	73.05	2	2	12	P
MCD	20110301	4:00:05	72.95	73.05	2	2	12	P

Table 6.2: This table represents trades for the stock 'MCD' on January 3rd, 2011. Due to preservation of data integrity, the true values for these times have been changed from their true values, and this figure is given only with the intent to show the reader the structure of the data.

shares. Similar to the 'COND' statistic, the 'MODE' statistic represents the type of quote. Once again, this is too detailed of a statistic for the purposes of our model. Therefore, we do not consider it in our work.

It falls to us to effectively aggregate and simplify this data into something that we wish to work with in our model. We recall that our algorithm given in 5 is two-fold. First, it decides at what times to trade, and then secondly, it defines how large of a trade to make at that time. Therefore, we must put this data into a form that allows us to observe market conditions at each second so that we can determine if that second is an ideal time in which to trade (t^*). This is the structure of the neural network model as given in 4.2.

6.1.2 Building the Neural Network Training Data from TAQ

As a result of the initial TAQ data, we glean from the data the following that we will use to build our neural network training data in order to properly locate ideal trading times. We glean from the TAQ data just listed the following statistics given in 6.1.2. We consider first the statistic given as 'MidPrice.' This is defined as the midpoint between the weighted bid and offer prices for each second. By weighted average we mean that for each time t

DateTime	MidPrice	BIDSIZ	OFRSIZ	RV	Price_Sim	Time	Trade
2011-01-19 09:34:18	75.051739	11.647059	4.058824	0.000014	74.73	258.0	0
2011-01-19 09:34:19	75.050000	10.857143	6.142857	0.000014	74.73	259.0	0
2011-01-19 09:34:20	75.050000	10.857143	6.142857	0.000014	74.73	260.0	0
2011-01-19 09:34:21	75.035000	8.000000	9.000000	0.000014	74.73	261.0	1
2011-01-19 09:34:22	75.040000	10.285714	1.714286	0.000014	74.73	262.0	0

Table 6.3: The this raw training data is gleaned from the initial TAQ data.

(seconds), we consider

$$\text{BID\_avg}_t = \frac{\sum_{i \in I_t} \text{BID}_i * \text{BIDSIZE}_i}{\sum_{i \in I_t} \text{BIDSIZE}_i} \tag{6.1}$$

$$\text{OFR\_avg}_t = \frac{\sum_{i \in I_t} \text{OFR}_i * \text{OFRSIZE}_i}{\sum_{i \in I_t} \text{OFRSIZE}_i} \tag{6.2}$$

where I_t is an index set the size of the number of quotes given at each second. This allows us to define the midprice as

$$\text{MidPrice}_t = \frac{1}{2}\left(\text{BID\_avg}_t - \text{OFR\_avg}_t\right) + \text{BID\_avg}_t. \tag{6.3}$$

We next consider both 'BIDSIZ' and 'OFRSIZ'. In order to realize these statistics we have to keep in mind that we are setting are midprice at time ti according to 6.3. Due to the nature of using the weighted averaging idea, the midprice may be a price that we do not have any quotes matching. Therefore, we cannot simply set the bid size and offer size via a direct match to the midprice. It is also the case that (unfortunate for our problem) there is a lot of intraday volatility with regards to the amount of volume available at the best bid and offer prices. This is for many reasons; the best bid and offer price is constantly changing, the volume available disappears when trades are made, and limit orders being canceled and added to the book at any time. Therefore, we simply define BIDSIZ (and similarly OFRSIZ) by the average bid size that is listed for time t.

$$\text{BIDSIZ}_t = \frac{\sum_{i \in I_t} \text{BIDSIZ}_i}{|I_t|} \tag{6.4}$$

$$\text{OFRSIZ}_t = \frac{\sum_{i \in I_t} \text{OFRSIZ}_i}{|I_t|} \tag{6.5}$$

The 'RV' statistic represents the realized volatility for the entire day that the model has calculated via 4.1 with $\tau = 5$ minutes (300 seconds) and T being the total amount of seconds in the trading day (normally $23,400$). Therefore, it is to be expexted that 'RV' is constant throughout the course of the entire training day. In order to fully utilize the definition given in 4.1, we need to note that X_t is the process for the midprice, MidPrice_t defined here with real data as 6.3,

$$RV_{(T,\tau)} = \frac{1}{T}\sum_{n=0}^{\frac{T}{\tau}} \left[\text{MidPrice}((n+1)\tau) - \text{MidPrice}(n\tau)\right]^2. \tag{6.6}$$

The 'Time' statistic means that this time is Time_t seconds from the beginning of the trading day. Therefore, if the time stamp is $9:30:00$ then this corresponnds to $\text{Time}_t = 0$ and $9:30:10$ implies that $\text{Time}_t = 10$. This allows the eventual trading algorithm to have a positional understanding of where the trader is according to the interval $[0, T]$ where T, in this case represents the end of the trading day.

The 'Trade' statistic is a binary indicator variable that represents from the 4.2, the 'TradeTick.' Thus, 'Trade' takes on a value of 1 if and only if The OFRSIZ statistic is greater than the mdeian offer size for the day, and the mid-price is in the bottom quartile of mid-prices for the day.

Of course, in a lot of the statistics just described, we had to look both backwards and forwards in time. This, as stated before is fine for our purpsoes at this point because we are still building adeqaute training data. We will see later on in this section, when we are actually training our model, that all of the inputs would be conceivable available to the trader using the algorithm at the time that the algorithm was implemented in real time. All of the statistics observed in 6.1.2 have been discussed other than 'Price_Sim.' This statistic is the one that is derived from the implementation of the Ogata thinning algorithm for multivariate Hawkes process simulations and is worthy of a more in depth discussion in order to see how it is obtained.

DateTime	RV	t	$\mathcal{O}$	$\mathcal{RM}$	$\mathcal{OD}$	$\mathcal{RMD}$	$\mathcal{H}$	TradeTick
2011-01-03 11:43:46	0.000017	7947.0	11.455556	76.880	0	0	0	0
2011-01-03 11:43:47	0.000017	7948.0	11.900000	76.880	0	0	0	0
2011-01-03 11:43:48	0.000017	7949.0	11.117391	76.880	0	0	0	1
2011-01-03 11:43:49	0.000017	7950.0	11.837391	76.878	1	1	0	1
2011-01-03 11:43:50	0.000017	7951.0	12.517391	76.874	0	1	0	1

Table 6.4: The initial calculations of the variables desired for neural network training in 4.2.

6.1.3 Using 4.1.1 to make 'Price_Sim' and Building the Neural Network Training Variables 4.2

In this section, we discuss the arrival of the 'Price_Sim' statistic from 6.1.2. Recalling how 4.1.1 functions, we need to obtain the MLE parameters $\hat{\mathbf{\Theta}}_{\text{MLE}}$ for the day from the microstructure movements of the data as related to the MidPrice Statistic. In implementation, we could use MLE's from the previous day that were to be representative of a process that lasts for two days in order to have a good indicator of the MLE's to use as inputs once our algorithm is trained and we are looking to implement it. However, our goal at the current juncture is building training data to train the algorith, and not to use it. We want our algorithm to be trained assuming that the thinning algorithm for multivariate Hawkes Processes is as accurate as it can be. This compels us to the conclusion that, for the purposes of training data, we wish to give $\hat{\mathbf{\Theta}}_{\text{MLE}}$ as defined by 4.11 where 4.14 and 4.15 are defined via N_1 and N_2 consisting of the actual arrivals of upward and downward movements of the midprice for that day. Thus, in a data sense, both N_1 and N_2 contain the times that upward and downward movements of the midprice take place, respectively during the day in question. From this point, we can find $\hat{\mathbf{\Theta}}_{\text{MLE}}$ via our constrained optimization method for 4.11 of choice. In order to achieve this, we use a technique known as "Sequential Quadratic Programming" described in [30]. We choose such a method because we are dealing with bounds in that for $\hat{\mathbf{\Theta}}_{\text{MLE}} = (\mu, \alpha, \beta)$, we need all elements $\mu, \alpha, \beta > 0$, and we also need to operate within the constraint that

$$\frac{\alpha}{\beta} > 1. \tag{6.7}$$

The algorithm works via use of Lagrange multipliers to find an adequate search direction to minimize the inputted function. After using this technique to obtain $\hat{\boldsymbol{\Theta}}_{\text{MLE}}$ for N_1 and N_2, we can use 4.1.1 to find the *simulated* arrival times of upward and downward movements of the mid-price. Initializing the process and the opening midprice for the day's data, we obtain a simulation for the mid-price, that we call 'Price_Sim.' This completes the description of the raw statistics that we use to build the variables for our neural network. We are now ready to build the statistics as they appear in the neural network training. We recall from 4.2, that the following variables need to be defined: t, $\mathcal{O}_t$, $\mathcal{OD}_t$, $\mathcal{RM}_t$, $\mathcal{RMD}_t$, $\mathcal{H}_t$, and $\mathcal{RV}_t$. Using the data from 6.1.2, we build the variables above based on their definitions to obtain the values found 6.1.3. These values are perfectly fine in terms of their calculations for the desired variables. However, they will not be well suited for the purpose of our artificial neural network model. The variables, in their current form require a bit of what is called "pre-processing." What we mean by "pre-processing" here is that we need to scale and transform some of the variables so as to make them more suitable for the ANN model. This is an important step when working with financial time series data (see [31], [32]). Pre processing is an important part of data science in that we need to ensure that all of our predictor data is equally weighted so that one data point does not carry so much weight throughout the model training phase. In order to see if any work is to be done, we do some exploratory data analysis on the variables as they are initially given in 6.1.3. The results are given in 6.1, and we can see that ther is a significant issue with the size of the variables. Realized volatility is much smaller than any other variable, while time and midprice are significantly larger. We wish to scale all of the continuous variables so that they are taking values in a similar window. We do this by upscaling smaller variables, and downscaling larger ones. For the time variable, we consider scaling the entire trading day into one unit

of time. We make the following transformations to our continuous variables.

$$t \to \frac{1}{23401}t \tag{6.8}$$

$$\mathcal{RM}_t \to \frac{1}{100}\mathcal{RM}_t \tag{6.9}$$

$$\mathcal{O}_t \to \frac{1}{100}\mathcal{O}_t \tag{6.10}$$

$$RV \to RV * 10000 \tag{6.11}$$

Applying these transformations to the continuous neural network variables, we obtain the distributions given in 6.2, which is far better in the sense that now our continuous features all take most of their values in essentially the same areas, yet we note that average offer size has some outlier values. In order to deal with the boolean variables, we use "one-hot encoding" in order to transform them into variables that our learning package knows to treat as categorical ones. At this point, it is time to fine tune our neural network, and choose the model that gives us the best measure of accuracy.

6.1.4 Training The Neural Network

As we have stated previously, we have used keras to build our neural network model. This allows us to train based on the rescaled values given in 6.1.3. We begin training the model judging for three characteristics throughout the process in loss, training accuracy, and validation accuracy. Loss, is the metric defined by the categorical cross entropy function that we have discussed in 2.26. Accuracy is a measure of how accurate the model is performing given as (via [33]) to be

$$A(y, y_{\text{true}})^n = \frac{1}{n}\sum_{i=0}^{n-1} \mathbb{1}_{y^i = y^i_{\text{true}}}. \tag{6.12}$$

The reason that we separate our metrics into training accuracy and validation accuracy is that we have split our data into two sets, training and test with the former being used to train the model. Training accuracy is a good sign as the model progresses in that it is learning based on its given data, but validation accuracy is an even better sign in that it portrays accuracy on data that the model has never seen during the training process. As

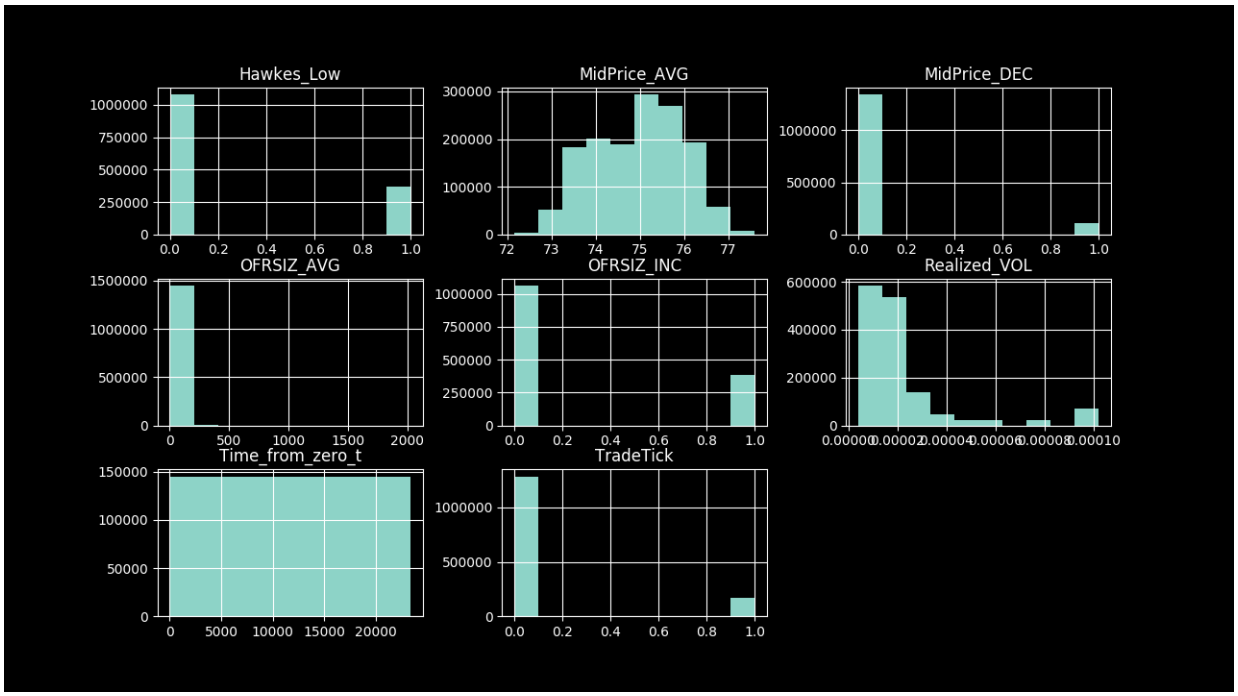

Figure 6.1: Distributions of neural network variables as given in 6.1.3.

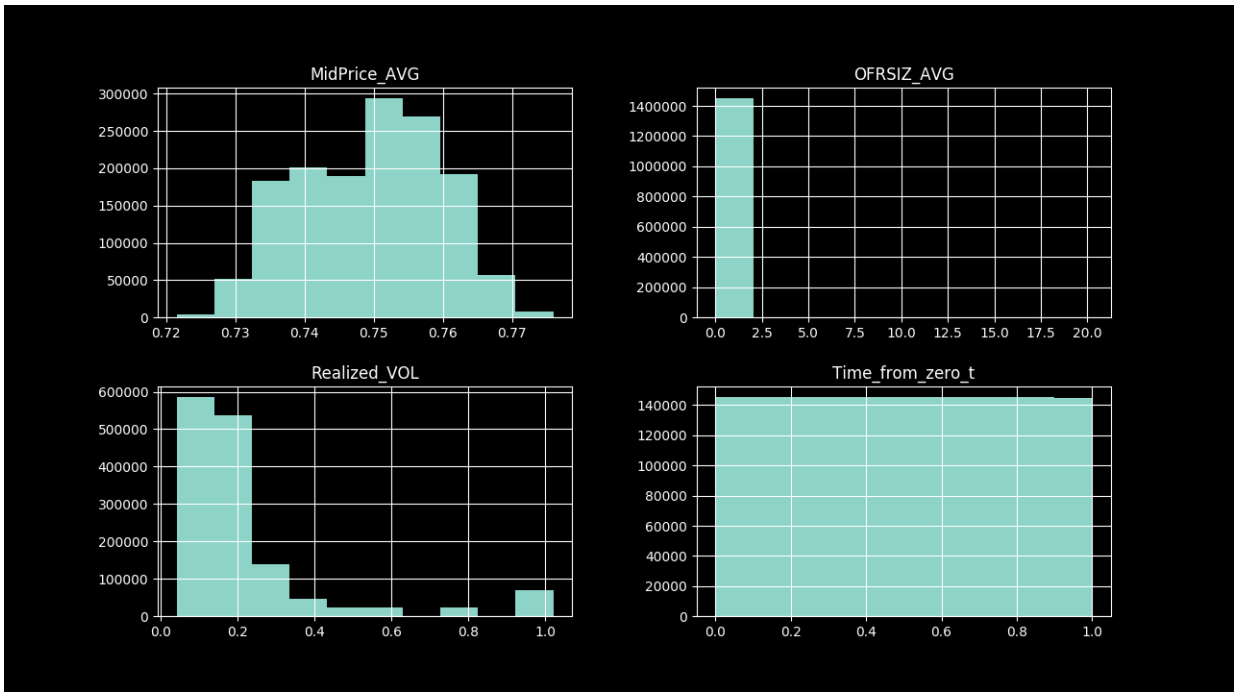

Figure 6.2: Distributions of neural network variables as given in 6.1.3.

Epoch	Loss	Training Accuracy	Validation Accuracy
145	.3797	.8280	.8283
146	.3798	.8281	.8322
147	.3797	.8282	.8348
148	.3795	.8284	.8308
149	.3793	.8285	.8208
150	.3793	.8283	.8244

Table 6.5: Progression of a Neural Network Model for MCD data Trade Tick classification

discussed before, with neural network models, a complete sweep through the data to train the model is called an "epoch." We train the MCD data with the variables given as the resampled values in 6.1.3 in a 3 layer artificial neural network using the ReLu activation function with the "adam" optimizer over 150 epochs to obtain the results for loss, training accuracy, and validation accuracy found in 6.5. This is a good sign that our neural network model is performing well on the data, and it is time to put it into use once fine tuned. A brief note on the training process itself is that in order to achieve this level of accuracy, we had to "upsample" the training data that was marked as a TradeTick by a ratio of 8 to 1. This is because initially, the Trade Ticks comprised only 12 percent of our data. This allows a deep learning algorithm to obtain a high degree of accuracy (in this 88 percent) even if it is marking every single entry as not a trade tick. Upsampling the training data was used to remedy this problem, and therefore our model has actually been trained on significantly more data points than we actually generated. We recognize that the scaling as indicated by 6.8 - 6.11 is a deviation from the standard method of standardizing and normalizing random variable data. This was taken into consideration as well, and the above method with 6.8 - 6.11 worked better for this data. To illustrate this, we consider the exact same neural network structure that produced the results given by 6.5, only this time, the continuous variables were pre-processed by a different technique, namely the standardization technique

Epoch	Loss	Training Accuracy	Validation Accuracy
195	.4986	.7427	.7437
196	.4988	.7427	.7491
197	.4985	.7428	.7438
198	.4983	.7428	.7405
199	.4986	.7426	.7454
200	.4986	.7427	.7451

Table 6.6: Progression of a Neural Network Model for MCD data Trade Tick classification

of

$$X \to \frac{X - \bar{X}}{X_{\max} - X_{\min}}. \tag{6.13}$$

This method, after 200 epochs in the same structure produced the results in 6.6. As one can see, these results are quite inferior to those of 6.5. In both of these situations, the loss is defined by the categorical cross-entropy loss function defined by 2.26.

6.1.5 The Reinforcement Learning Algorithm Implementation

We can tweak the Policy improvement algorithm 3 once we have calculated the transition matrix based on the same data (appended with the trades) that has been used to train the neural network model using 4.37. The parameters that we can use to tweak the transition matrix towards different outcomes are q_0, the number of shares that the trader must purchase, and $\mathcal{L}_{\text{high}}$, the number of shares that are purchased in order for the action of 'Buy High' to be true. This will alter the course of the trader in that if $\mathcal{L}_{\text{high}}$ is low, then the trader will often learn that making high volume market orders is not too much of a risk. With different values of q_0, the trader will find differing degrees of probability in terms of leaving and entering the purchase remaining states $X_1, X_2,$ and TERM. For instance, if q_0 is very high, then it is the case that the probability of any given trade taking the trader

$\mathcal{L}_{\text{high}}$	$\lvert q_0 \rvert$	Reward	Optimal Policy
8000	100000	$\{r_{\text{match}} = 3,\ r_{\text{slip}} = -5,\ r_{\text{shock}} = 10\}$	(X1,t1): high, (X1,t2): high, (X1,t3): high, (X2,t1): high, (X2,t2): high, (X2,t3): low
8000	100000	$\{r_{\text{match}} = 3,\ r_{\text{slip}} = -30,\ r_{\text{shock}} = 8\}$	(X1,t1): low, (X1,t2): high, (X1,t3): high, (X2,t1): low, (X2,t2): high, (X2,t3): low
15000	50000	$\{r_{\text{match}} = 8,\ r_{\text{slip}} = -5,\ r_{\text{shock}} = 4\}$	(X1,t1): low, (X1,t2): low, (X1,t3): low, (X2,t1): low, (X2,t2): low, (X2,t3): low

Table 6.7: Tuning of the reinforcement learning parameters giving different results for

to the next purchase state will be quite low. If q_0 is rather low, this probability will be higher. We also have at our disposal the ability to change the trader's reward vector. A low value of r_{slip} and a high value of r_{shock} will make the trader very aggressive with placing large market orders in that there won't be much penalty for walking the book, and the reward for matching a large trade is really high. We list some combinations in 6.7 that illustrate this. In order to run our final simulation, we choose the parameters to be (X1,t1): low, (X1,t2): high, (X1,t3): high, (X2,t1): low,(X2,t2): high, (X2,t3): low in order to provide a grounded nature of our trader by keeping the reward for walking the book large and negative, as well as some incentive for being more aggressive by keeping the reward for large orders high. This gives our simulated optimal policy as above. For the remaining parameters, we set $\mathcal{L}_{\text{high}} = 8000$ and $|q_0| = 100000$.

6.1.6 Running the Complete Trading Algorithm

We use the optimal policy given by

(X1,t1): low, (X1,t2): high, (X1,t3): high, (X2,t1): low,(X2,t2): high, (X2,t3): low (6.14)

as calculated from a transition matrix defined by $\mathcal{L}_{\text{high}} = 8000$ and $|q_0| = 100000$. From here, we run our trained neural network model to locate times that are good to trade, and then trade based on the state and optimal policy. For a position in which $|q_0| = 100000$, and $\mathcal{L}_{\text{low}} = 1000$,the algorithm performs the purchase market orders as indicated by 6.1.6.

One notices that since our trader can execute a trade uniquely at any second he wishes, that with a relatively small amount that he must unwind, it is often done quickly. In order to keep the threshold of accuracy for locating the trading ticks high, the probability threshold of the model was increased. It remains to be seen how much money the trader has saved as a result of using this model. In order to do this, we look to compare it with the HJB approach.

6.1.7 Comparison with the HJB approach

In order to evaluate how well our model has performed, we need to consider its performance against an established model. To this end, we consider the model given in section 6.3 of [3] that is defined for the situation defined as "Optimal execution using market orders without penalties and only temporary impact."

The model makes the following assumptions. The first is that the agent's own trades have no effect on the midprice of the asset. This is consistent with our data based approach. If we did not make this assumption, then both our neural network algorithm and reinforcement learning algorithm would be in trouble. For instance, our neural network has scanned the path of midprice movements to best predict optimal trading midprice points based on the trades that were occurring in the market. If, our trades at the points it recognized were to effect the midprice behavior of the market, then the path would be completely thrown off. The neural network model did not account for this.

With respect to the reinforcement learning algorithm, it should be noted that the reward structure of the reinforcement learning algorithm includes rewards for the the agent's trades matching, walking the book, and matching the book with a large volume. It is important to recognize that if a result of our trades included moving the entire midprice of the market, and not just walking the book, that any reincforcement learning model should account for such an action. If this were possible, and our algorithm did not account for it, then our trader could very easily find a strategy in which midprice movements were not taken into account by his actions, leaving all subsequent trades after the first open open to a large amount of price risk that had never been accounted for.

$\|q_t\|$	Time ($[0,1]$)	State	Policy	Amount Purchased	$\|q_{t+1}\|$
100000.000000	0.060425	(X1,t1)	low	3315.925590	96684.074410
96684.074410	0.060510	(X1,t1)	low	7303.094454	89380.979956
89380.979956	0.062690	(X1,t1)	low	6231.379817	83149.600140
83149.600140	0.062732	(X1,t1)	low	5073.066020	78076.534119
78076.534119	0.062775	(X1,t1)	low	5620.387271	72456.146849
72456.146849	0.062818	(X1,t1)	low	3622.916461	68833.230388
68833.230388	0.062861	(X1,t1)	low	7905.611900	60927.618488
60927.618488	0.062903	(X1,t1)	low	4286.294070	56641.324417
56641.324417	0.062946	(X1,t1)	low	3912.717238	52728.607179
52728.607179	0.062989	(X1,t1)	low	1358.877044	51369.730135
51369.730135	0.063031	(X1,t1)	low	2078.037261	49291.692874
49291.692874	0.063074	(X2,t1)	low	4055.030800	45236.662074
45236.662074	0.063117	(X2,t1)	low	7226.242256	38010.419818
38010.419818	0.063160	(X2,t1)	low	6976.396468	31034.023350
31034.023350	0.063202	(X2,t1)	low	6455.801709	24578.221641
24578.221641	0.063245	(X2,t1)	low	2277.579262	22300.642379
22300.642379	0.063288	(X2,t1)	low	5648.680243	16651.962135
16651.962135	0.063331	(X2,t1)	low	4701.718429	11950.243706
11950.243706	0.063373	(X2,t1)	low	4557.722094	7392.521612
7392.521612	0.063416	(X2,t1)	low	3320.080301	4072.441311
4072.441311	0.064869	(X2,t1)	low	1192.690929	2879.750383
2879.750383	0.064912	(X2,t1)	low	6000.985396	-3121.235013
-3121.235013	0.065467	(TERM,t1)	NONE	0.000000	-3121.235013
-3121.235013	0.065510	(TERM,t1)	NONE	0.000000	-3121.235013
-3121.235013	0.065553	(TERM,t1)	NONE	0.000000	-3121.235013

Table 6.8: The Neural Network Reinforcement Trading Algorithm applied to MCD on 3/7/2011 data.

Another assumption from the the HJB model in [3] is that the agent's trades have a temporary price impact in their own execution. What is meant by this is that when the agent trades, without effecting the midprice, his trades can walk the book and pay a worse price based on walking the limit order book. This is perfectly in line with our reinforcement learning algorithm which issues a negative reward to the trader for precisely this action. Also, the neural network model is trained to recognize 'TradeTicks' that have high best price volume avialable in order to recognize optimal times at which to trade.

The next assumption that this model makes, that at first will seem different from our model is that the bid ask spread is equal to zero, and thus all trades are conducted assuming the midprice will be the best point of execution. To this end, we note that our model, though trained to find times where the offer price has high liquidity, is still tracking all of the optimal trading times based purely on movements of the midprice. This is the case in both the historical path of the price as well as the path of the Hawkes Process simulation that have been inputted into the neural network model. Therefore, this assumption fits fine within the confines of our model.

The final assumption of the HJB model is that all of the order is executed at the end of the time limit, that is by time T. This has is a key assumption of the initial framework of our model.

6.1.7.1 The HJB Model for Optimal Execution of Market Orders with Temporary Impact

The HJB model from [3] is set up as follows. The goal is to acquire $\mathcal{N}$ shares by time T. We consider the control process of the trader to be the trading curve $(v_t)_{\{0\leq t\leq T\}}$. This is the speed at which the agent is purchasing/liquidating shares per unit time. The agent's inventory is considered to be $(Q_t^v)_{\{0\leq t\leq T\}}$. This inventory represents the number of shares that are in the trader's posession at time t. It is going down over the interval $[0,T]$ in a liquidation problem and increasing from $[0,T]$ for an acquisition problem. The midprice process is given such that the dynamics are

$$dS_t^v = \pm g(v_t)dt + \sigma dW_t \tag{6.15}$$

$$S_0^v = S_0 \tag{6.16}$$

We note that in 6.15, that $\pm g(v_t)$ represents the permanent price impact of of the model. In this instance, $g(v_t) = 0$ due to the no permanent price impact via our trading process (v_t) assumption. Here, *sigma* represents a volatility parameter, and W_t a Brownian motion process. S_0 represents the initial midprice of the equity at time 0.

Two other processes considered are $\hat{S}_t^v$, and X_t^v, which represent the price paid by the agent at time t, and the cash process of the agent at time t respectively. The dynamics of the inventory are given by

$$dQ_t^v = \pm v_t dt \tag{6.17}$$

$$Q_0^v = q_0. \tag{6.18}$$

Thus, the inventory is effected only by the trading activity, and the initial inventory is the position that the trader is attempting to unwind. The dynamics of the trader's price have the most interesting dynamics in that

$$\hat{S}_t^v = S_t^v + (\frac{1}{2}\Delta + f(v_t) \tag{6.19}$$

$$\hat{S}_0^v = \hat{S}_0. \tag{6.20}$$

In 6.19, $f(v_t)$ represents the temporary price impact based on the trader's control process at the time that the trade was made. This model assumes that such a temporary impact function is linear in the sense that $f = kv_t$ for some constant k.

The expected cost for the trader acquiring these shares, given a terminal penalty α is given by

$$EC^v = \mathbb{E}\left[\int_t^T \hat{S}_u^v v_u du + ((N) - Q_T^v)S_T + \alpha(\mathcal{N} - Q_T^v)^2\right]. \tag{6.21}$$

This gives the value function defined to be

$$H(t,S,y) = \inf_{v\in\mathcal{A}} \mathbb{E}_{t,S,y}\left[\int_t^T \hat{S}_u^v v_u du + y_T^v S_T + \alpha(Y_T^v)^2\right], \tag{6.22}$$

where $Y_t^v = \mathcal{N} - Q_t^v$. This gives the optimal acquisition rate to be

$$v_t^* = \frac{\mathcal{N}}{T + \frac{k}{\alpha}}.$$

6.1.8 A comparison of our model and the HJB Model

As one can imagine, a key advantage of our model over the HJB model is that our model can handle trading larger amounts since it has been trained to recognize when walking the book is unlikely, and make large trades. Therefore, we expect that our model will handle market risk (risk that the market changes) much better than the HJB model. This turns out to be the case. Consider a day when the price gets higher throughout the day. The HJB model will trade an equal number of shares every second, but our neural network based reinforcement learning algorithm will locate better times to place large orders throughout the day.

We ran this model on a day where the market shifted throughout the day, no doubt exposing the trader to considerable market risk. The results can be seen in 6.1.8. In order to compare our models the way that the HJB models do, using implementation shortfall, we note that the benchmark price for this day was \$72.41. The HJB strategy's average price came in at \$73.63. The average price per share based on using the Neural Network Reinforcement Learning model was \$73.06. This result beats the HJB approach on the implementation shortfall scale by about 57 cents per share.

6.1.9 Comparison With a More Advanced HJB Model

The linear HJB model is a rather simple one. We wish to see how our model performs up against a model that is essentially designed for a situation in which the trader is facing considerable market risk. To this end we consider the model from section 7.2 of [3] for optimal acquisition with a price Limiter.

This model considers the case of a trader who wishes to acquire a number of shares $\mathcal{N}$ by time T. This model considers the same overall market dynamics of the previous linear model, namely 6.15, 6.17, and 6.19. However, we add a new layer to this model in that the trader will stop trading not only if his inventory hits the terminal level (all desired shares have been acquired); $q_t = \mathcal{N}$. Strategic trading also stops if the price hits an upper threshold. That is, if at any time $t \in [0, T]$, $S_t^v = \bar{S}$, then the strategy will cease to be operational, and a market order will be placed for the execution of the remaining inventory. The same action

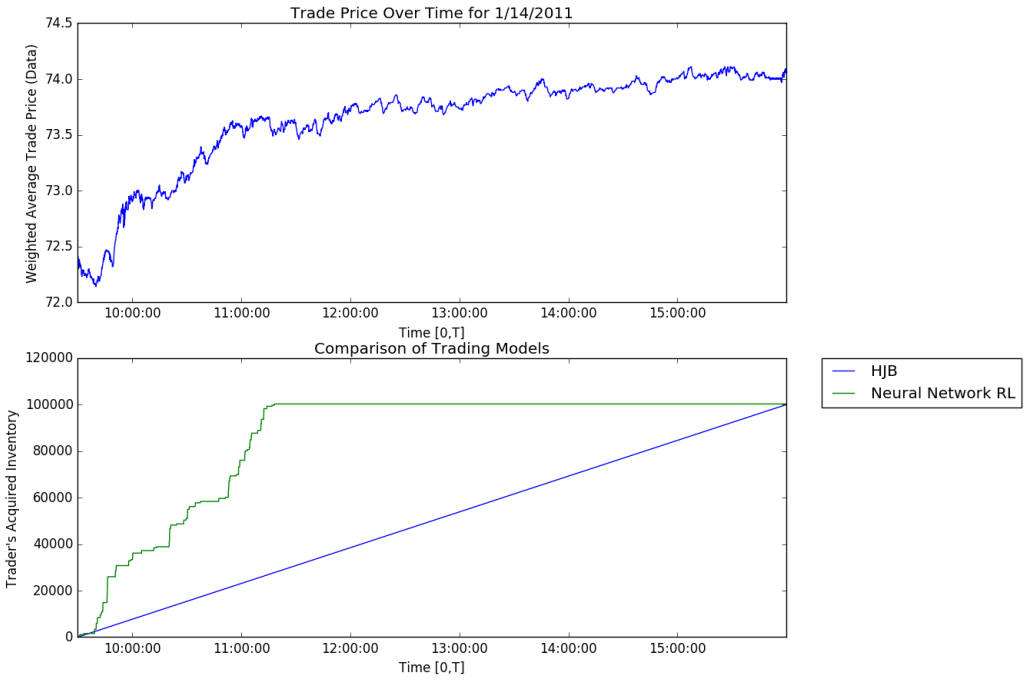

Figure 6.3: This curve shows the difference betweeen the two strategies on a day where there is considerable market risk. Here, $\alpha = 100k$, and $k = 10^{-4}$

also happens if time T is reached before the entire desired inventory has been acquired. Therefore, this model has a stopping time τ for which

$$\tau = T \wedge \inf t : S_t = \bar{S} \wedge t : Q_t = 0. \tag{6.23}$$

At a stopping time τ, the agent pays

$$S_\tau + \alpha(\mathcal{N} - Q_\tau^v) \tag{6.24}$$

per share. The agent realizes the amount of inventory left to be purchased via the process

$$y_t^v = \mathcal{N} - Q_t^v. \tag{6.25}$$

This gives rise to the value function for this problem of

$$\mathbb{E}_{t,S,y}\left[\int_t^\tau (S_u + kv_u)v_u du + y_\tau(S_\tau + \alpha y_\tau) + \phi \int_t^\tau y_u^2 du\right]. \tag{6.26}$$

In 6.26, $\int_t^\tau (S_u + kv_u)v_u du$ represents the price paid per share before the strategy has reached a stopping time. $y_\tau(S_\tau + \alpha y_\tau)$ represents the amount paid per share at a stopping time, with α representing the penalty that the trader will pay for executing a large market order. $\phi \int_t^\tau y_u^2 du$ represents the penalty imposed on a trader for not executing inventory positions in a timely manner. Thus, a penalty is assessed at each moment in time based on the amount of shares the agent still has to acquire. Given this framework, the problem becomes to find an admissible strategy v_t such that

$$H(t, S, y) = \inf_{v \in \mathcal{A}} H^v(t, S, y). \tag{6.27}$$

The dynamic programming equation for 6.27

$$\partial_t H + \frac{1}{2}\sigma^2 \partial_{SS} H + \phi y^2 + \min\{-v\partial_y H + bv\partial_S H + (S + kv)v\} = 0 \tag{6.28}$$

$$H(T, S, y) = (S + \alpha y)y \tag{6.29}$$

$$H(t, \bar{S}, y) = (\bar{S} + \alpha y)y \tag{6.30}$$

$$H(t, S, 0) = 0 \tag{6.31}$$

Due to the nature of our assumptions, we once again do not include a model for permanent price impact. Thus, we set $b = 0$, and after a dimensionality reduction (in y) this pde becomes

$$\partial_t h + \frac{1}{2}\sigma^2 \partial_{SS} h - \frac{1}{k}h^2 + \phi = 0 \tag{6.32}$$

$$h(T, S) = \alpha \qquad S \leq \bar{S} \tag{6.33}$$

$$h(t, \bar{S}) = \alpha \qquad t \leq T, \tag{6.34}$$

where the optimal trading curve is then given as

$$v^*(t, S, y) = \frac{1}{k} y h(t, S). \tag{6.35}$$

Lacking an explicity solution for 6.32, we resort to a Crank-Nicolson scheme as described in [34] to solve it numerically. In order to do this, we need to place the problem onto a grid. This requires us to impose a lower price boundary $\underline{S}$, givin the grid $[0, T] \times [\underline{S}, \bar{S}]$. Setting

$$\chi(t) = h(t, \underline{S}), \tag{6.36}$$

then along this lower boundary condition, χ can be explicitly solved giving the solution

$$\chi(t) = \begin{cases} \sqrt{k\phi}\frac{\zeta e^{2\gamma(T-t)}+1}{\zeta e^{2\gamma(T-t)}-1} & \phi > 0 \\ \left(\frac{1}{\alpha} + \frac{T-t}{k}\right)^{-1} & \phi = 0 \end{cases} \tag{6.37}$$

$$\gamma = \sqrt{\frac{\phi}{k}} \tag{6.38}$$

$$\zeta = \frac{\alpha + \sqrt{k\phi}}{\alpha - \sqrt{k\phi}}. \tag{6.39}$$

The discretization of the equation 6.32 can now begin. We note that we have boundary conditions present for all but the boundary $h(0, S)$. Therefore, we wil discretize for the purpose of the Crank-Nicolson algorithm going backwards in time beginning at the boundary T. There is also the matter of the nonlinear term

$$\frac{1}{k}h^2.$$

In order to effectively discretize this term for the purposes of Crank-Nicolson, we will lag it back one time step and treat it explicitly. Let $h(t_n, S_i) = h_i^n$, and discretize with respect to time and stock price via a centered difference scheme to obtain

$$\frac{h_i^n - h^{n-1}}{\Delta t} + \frac{1}{2}\sigma^2 \left(\frac{1}{2} \left(h_{SS_i}^{n-1} + h_{SS_i}^n \right) \right) - \frac{1}{k}(h_i^n)^2 + \phi = 0$$

$$\frac{h_i^n - h^{n-1}}{\Delta t} + \frac{\sigma^2}{4(\Delta x)^2} \left[h_{i+1}^{n-1} - 2h_i^{n-1} + h_{i-1}^{n-1} + h_{i+1}^n - 2h_i^n - 2h_i^n + h_{i-1}^n \right] - \frac{1}{k}(h_i^n)^2 + \phi = 0.$$

Multiplything through by $4\Delta t(\Delta x)^2 k$, we obtain

$$(h_i^n - h_i^{n-1})4(\Delta x)^2 k + \sigma^2 \Delta t k \left[h_{i+1}^{n-1} - 2h_i^{n-1} + h_{i-1}^{n-1} + h_{i+1}^n - 2h_i^n + h_{i-1}^n \right] - \tag{6.40}$$

$$4\Delta t(\Delta x)^2 (h_i^n)^2 + 4\Delta t(\Delta x)^2 k\phi = 0. \tag{6.41}$$

From 6.40, we can put the problem into a tridiagonal matrix to solve, at each backward time step:

$$\begin{pmatrix} B_0 & C_0 0 & 0 & 0 & 0 \\ A_1 & B_1 & C_1 & 0 & 0 \\ 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & \cdot & B_{I-1} & C_{I-1} \\ 0 & 0 & 0 & A_I & B_I \end{pmatrix} \begin{pmatrix} h_0^{n-1} \\ h_1^{n-1} \\ \cdot \\ \cdot \\ h_I^{n-1} \end{pmatrix} = \begin{pmatrix} D_0 \\ D_1 \\ \cdot \\ \cdot \\ D_I \end{pmatrix} \tag{6.42}$$

Where, via rearranging 6.40, we obtain the coefficients

$$A_i = \sigma^2 \Delta t k \tag{6.43}$$

$$B_i = -2\sigma^2 \Delta t k - 4(\Delta x)^2 k \tag{6.44}$$

$$C_i = \sigma^2 \Delta t k. \tag{6.45}$$

The right hand side of this equation is given by

$$D_i = -4(\Delta x)^2 k u_i^n - \sigma^2 \Delta t k u_{i+1}^n + 2\sigma^2 \Delta t k u_i^n - \sigma^2 \Delta t k u_{i-1}^n + 4\Delta t(\Delta x)^2 (u_i^n)^2 - 4\Delta t(\Delta x)^2 k\phi. \tag{6.46}$$

The boundary conditions are substitued for D_0 and D_I with $B_0 = 1,\ C_0 = 0, A_I = 0, B_I = 0$ to complete the matrix at each time step. Plugging in values for k, α, and σ, we can now solve the problem.

Fitting the problem to our data and using the resources of [35], we can obtain the results in 6.1.9, and see that our model performs worse overall, since the Crank-Nicolson model obtains an average price per trade of \$72.29. We do note that this model is designed for precisely the situation as indicated in the data. Thus it remains to be seen how well machine learning models can adequately be improved the beat the best model for each situation.

6.2 SIMULATIONS FOR THE LIMIT ORDER MODEL

6.2.1 An Adequate model for comparison

We seek to compare this model to a model from [3] for optimal execution using only limit orders. In this model, we consider a trader who has $\mathcal{N}$ shares that need to be liquidated by time T beginning at time $t = 0$. This model considers the price dynamics where (no permanent impact from the trader) the midprice is driven by

$$S_t = S_0 + \sigma W_t. \tag{6.47}$$

The trader, at each moment in time t, is constantly canceling outstanding orders and placing new limit orders at a position δ_t. δ_t defines the limit order in the sense that the limit order is placed at the position

$$S_t \pm \delta_t. \tag{6.48}$$

The position 6.48 is defined with a positive sign on δ if the goal of the problem is to liquidate, and negative sign on δ if the goal is to acquire.

One thing to note here is that a limit order (of a reasonably small size in volume) that is placed with too small of a $delta_t$ behaves essentially like a market order. To see this, consider that in practice,

$$\underline{S_t} < S_t < \overline{S_t},$$

where $\underline{S_t}$ and $\overline{S_t}$ are the price of the best bid, and best ask price respectfully at time t. This means that if $\delta_t < (\overline{S_t} - S_t)$, then the limit order will actually be placed at a better position

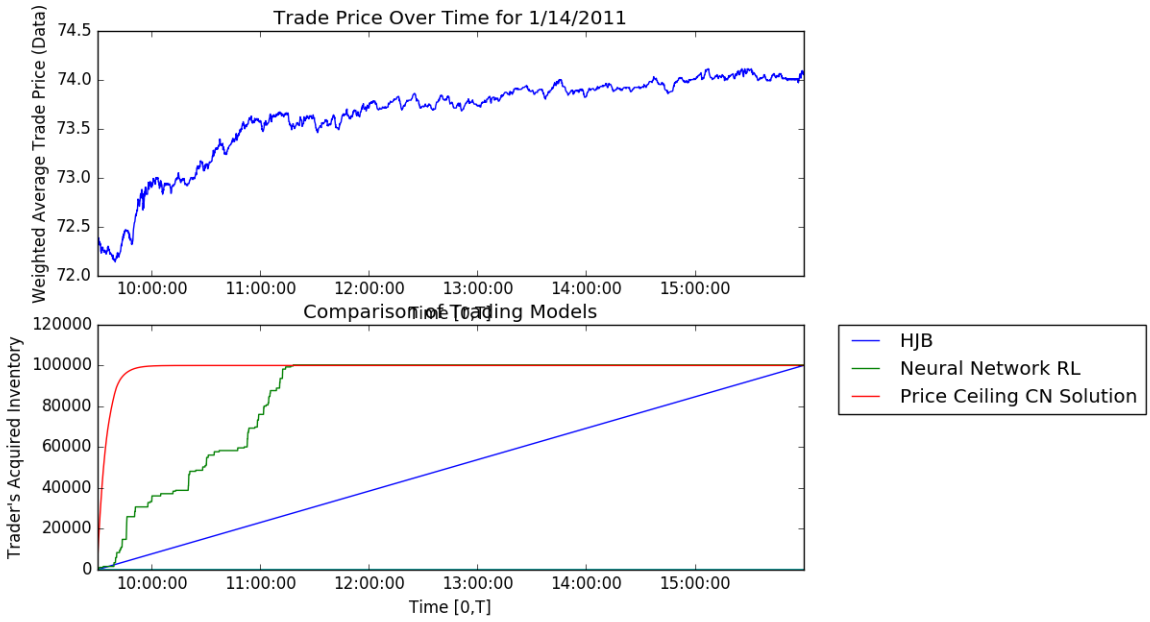

Figure 6.4: The inventory and price of each execution model. Here, $\alpha = 100k$, and $k = 10^{-4}$ for both HJB models.

than the currest best bid price. This, means that the second a matching size market buy order enters the market, that it will execute the order placed at a small δ_t upon arrival.

When dealing with limit orders, a key difference from market order placement is that the trader does not have as much control over the execution. He must wait until an opposing market order (or in some rare cases a matching limit order) comes in to the market to buy/sell his position. Since the trader does not have control of the overall order flow of the market, such actions must be considered in a random process.

To this end, for the optimal liquidation problem, let M_t for $0 \leq t \leq T$ denote a Poisson process (with intensity λ) corresponding to the counting process of market buy orders that have arrived in the market by time t. This, though important is not the real process that the agent is most interested in. The agent's chief concern is the counting process defined by

$$N_t^\delta = (N_t^\delta)_{0 \leq t \leq T}.$$

This process N_t^δ is the arrival process of market orders that **lift the order placed at** δ_t. For instance, if $\delta = \$.04$, and is for a size of 400 shares, then N_t jumps if and only if a market buy order enters the market for more than 400 shares (or what is remaining) and $S_t + .04$ is now the best price in the book. The entrance of this market order will also make M_t jump, but so will every single market buy order. The nature of these two processes motivates the probability function

$$P(\delta) = e^{-\kappa\delta}. \tag{6.49}$$

This linear execution function has been defined before as the probability that the next incoming market order lifts the placed limit order. In other words, this is the probability that the next arrival of M_t will also be an arrival of N_t^δ.

The agent's cash process X_t has different in a liquidation problem as opposed to an acquisition problem in that he acquires cash at every step of the liquidiation as opposed to using it to purchase shares. It therefore has the dynamics given by

$$dX_t^\delta = (S_t + \delta_t) dN_t^\delta. \tag{6.50}$$

The final process considered in the model is given as the inventory. That is, the position that remains to be liquidated:

$$Q_t^\delta = \mathcal{N} - N_t^\delta. \tag{6.51}$$

The trader then seeks the optimal solution for the optimization function

$$H(x,S) = \sup_{\delta\in\mathcal{A}} \mathbb{E}\left[X_\tau^\delta + Q_\tau^\delta(S_\tau - \alpha Q_\tau^\delta)|X_{0^-}^\delta = x,\ S_0 = S, Q_{0^-}^\delta = \mathcal{N}\right]. \tag{6.52}$$

In 6.52, $\tau = T \wedge \min\{t:\ Q_t = 0\}$ and α represents a penalty parameter for holding inventory at liquidation. The resulting DPE can be solved giving the optimal limit order placement to be

$$\delta^*(t,q) = \frac{1}{\kappa}\left[1 + \log\frac{\sum_{n=0}^{q}\frac{\tilde{\lambda}^n}{n!}e^{-\kappa\alpha(q-n)^2}(T-t)^n}{\sum_{n=0}^{q-1}\frac{\tilde{\lambda}^n}{n!}e^{-\kappa\alpha(q-1-n)^2}(T-t)^n}\right]. \tag{6.53}$$

In 6.53, $\tilde{\lambda} = \lambda e^{-1}$, and q represnents $Q_t^{\delta^*}$.

6.2.2 Algorithm 6 for the case of constant re-evaluation of outstanding orders

In the case of 6, it was the case that the orders would sit in the book, and the inventory would simply be updated. We can also adjust 6 for the case that the orders are re-evaluated every 5 seconds. In this case, the 'Volume Executed' will simply update assuming that the orders are canceled, and this poses no major change. The only major change, is that the structure of the training input τ for 5.4 is no longer really a variable. Intead it is fixed at this interval in which we are constantly checking for cancellation (say $\tau = 5$ seconds), and is no longer a variable worth training on since it is now constant. Other than that, the algorithm will run as it is written in 6.

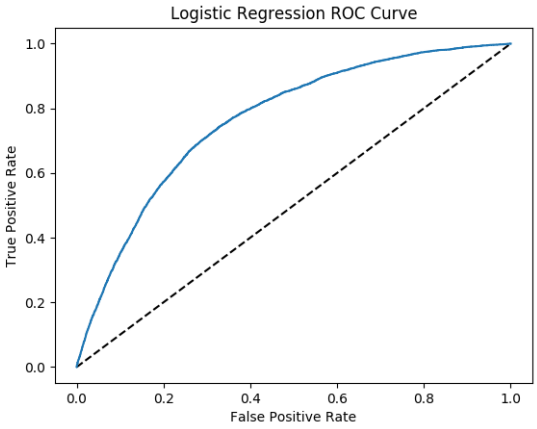

Figure 6.5: The ROC curve for the probability function logistic regression method.

6.2.3 A comparison of the two models

The model in 6 is run on 60 seconds worth of data alongside the standard HJB model. The goal of the trader in this case is to optimally liquidate 1000 shares. The HJB model operates by placing 100 volume limit orders every second, and will re-evaluate the position and cancel any outstanding orders every second. Our model, gives a little bit more leniency, and will allow orders to remain in the limit order book for a total of five seconds. While training our limit order probability function, we obtain a score of the model that is fair, one of a 76.25 roc-auc (area under the receiver operator characteristic curve). It is shown for this exact model run in 6.2.3.

Once the probability function has been trained the optimal limit order placements are calculated via optimizing the utility function. From here, we can calcuate our optimal trading curve, as well as the optimal trading curve of the HJB model via 6.53 to obtain the results shown in 6.6. We note that even though the price that the shares are sold for in the HJB model is higher (about $77.015 cents as opposed to about $77.00), the HJB model is unable to complete the trade. The logistic regression model makes its optimal placement at the midprice because of the reasonably low intra-second volatility, we see in 6.6, and thus is able

to execute the entire volume in one minute, costing the trader little more than one cent per share, and without having to deal with the large terminal time execution penalty, which for the HJB model is that for terminally liquidating via a market sell order, 800 of the desired 1000 shares.

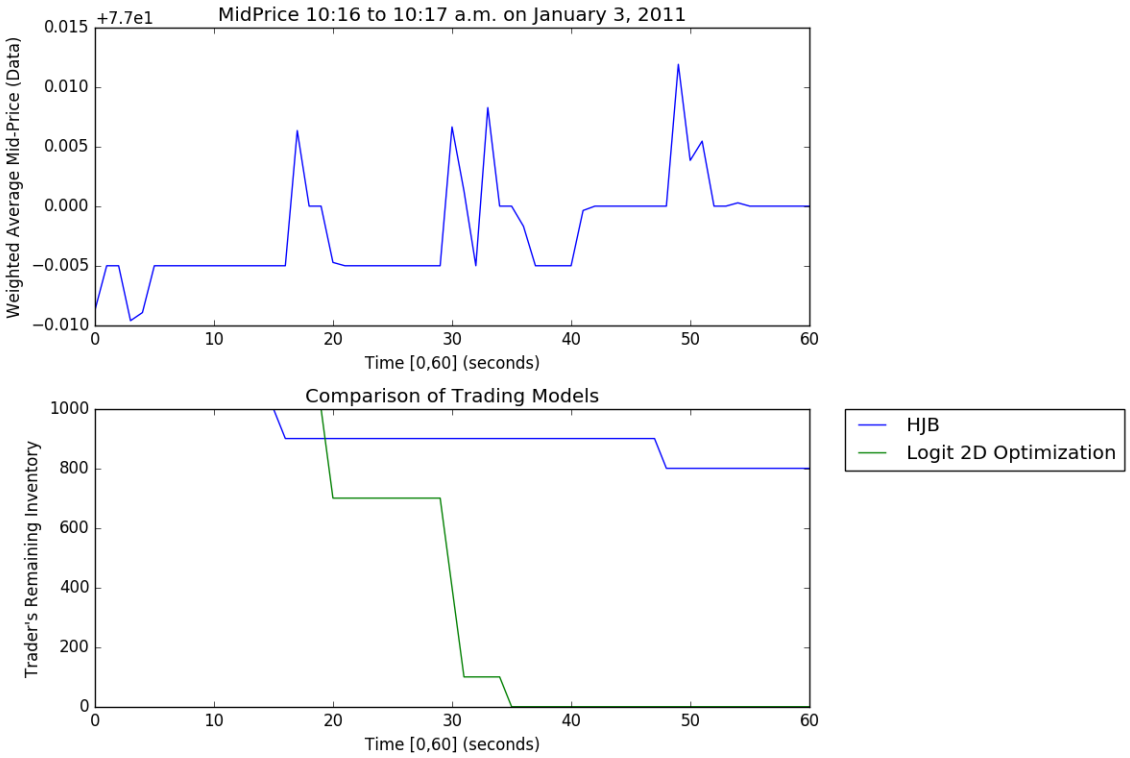

Figure 6.6: A comparison of the two models on 60 seconnds of MCD data. For the HJB Model, $\lambda = 158$, $T = 60$, $k = 100$, $\alpha = .001$, and $\mathcal{N} = 1000$.

7.0 CONCLUSIONS

7.1 SUMMARY OF RESULTS

This research has given rise to a two stage optimal execution model for market orders. This model, using historical market data is able to produce an optimal purchase trading curve via locating times in stage one at which to trade via an artificial neural network model (ANN). In stage two, the model decides how much to trade at these times via a reinforcement learning algorithm.

This method could be redone to include multiple equities, and can be easily adjusted to produce a trading curve for optimal liquidiation.

Our research also produced an optimal trading curve for optimal liquidation using only limit orders using logistic regression to train the probability of limit order execution.

We hope that as the world enters into the exiting an innovative age that will show what advancements can be made via machine learning, that there will be thought to how it can be used in conjunction with established mathematics to produce further results in financial market modeling.

7.2 FUTURE WORK

We most wish to improve the market order algorithm. This will begin with enhancing the Hawkes process model so that it accounts for intraday volatility at various times of the day, giving the trader a better idea of when to make a move.

Another aspect we wish to improve upon is the reinforcment learning algorithm itself to make it incorporate more states, giving a more accurate picture of the market.

Towards the machine learning end of things, we wish to keep experimenting with more advanced neural network structures in order to obtain the one best fit for financial time series data.

8.0 BIBLIOGRAPHY

[1] Yuanda Chen. Multivariate hawkes processes and their simulations. 2016.

[2] Olivier Gu´eant. *The Financial Mathematics of Market Liquidity: From optimal execution to market making*, volume 33. CRC Press, 2016.

[3] ´Alvaro Cartea, Sebastian Jaimungal, and Jose´ Penalva. *Algorithmic and high-frequency trading*. Cambridge University Press, 2015.

[4] Dimitris Bertsimas and Andrew W Lo. Optimal control of execution costs. *Journal of Financial Markets*, 1(1):1–50, 1998.

[5] Robert Almgren and Neil Chriss. Value under liquidation. *Risk*, 12(12):61–63, 1999.

[6] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40, 2001.

[7] Robert F Almgren. Optimal execution with nonlinear impact functions and trading-enhanced risk. *Applied mathematical finance*, 10(1):1–18, 2003.

[8] Jim Gatheral. No-dynamic-arbitrage and market impact. *Quantitative finance*, 10(7):749–759, 2010.

[9] Robert Almgren. Optimal trading with stochastic liquidity and volatility. *SIAM Journal on Financial Mathematics*, 3(1):163–181, 2012.

[10] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[11] D Stathakis. How many hidden layers and nodes? *International Journal of Remote Sensing*, 30(8):2133–2147, 2009.

[12] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[16] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[17] Wojtek J Krzanowski and David J Hand. *ROC curves for continuous data*. CRC Press, 2009.

[18] Oleh Danyliv, Bruce Bland, and Daniel Nicholass. Convenient liquidity measure for financial markets. 2014.

[19] Aurélien Alfonsi and Pierre Blanc. Dynamic optimal execution in a mixed-market-impact hawkes price model. *Finance and Stochastics*, 20(1):183–218, 2016.

[20] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[21] US Securities, Exchange Commission, et al. Final rule: Disclosure of order execution and routing practices, 2000.

[22] Robert Almgren. High frequency volatility. *New York University. Google Scholar*, 2009.

[23] Emmanuel Bacry, Sylvain Delattre, Marc Hoffmann, and Jean-François Muzy. Modelling microstructure noise with mutually exciting point processes. *Quantitative Finance*, 13(1):65–77, 2013.

[24] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[25] Yosihiko Ogata. On lewis' simulation method for point processes. *IEEE Transactions on Information Theory*, 27(1):23–31, 1981.

[26] Hans R Stoll. The supply of dealer services in securities markets. *The Journal of Finance*, 33(4):1133–1151, 1978.

[27] Dieter Hendricks and Diane Wilcox. A reinforcement learning extension to the almgren-chriss framework for optimal trade execution. In *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2104 IEEE Conference on*, pages 457–464. IEEE, 2014.

[28] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680. ACM, 2006.

[29] WRDS. Nyse trade and quote. https://wrds-web.wharton.upenn.edu/wrds/, 2011.

[30] Stephen Wright and Jorge Nocedal. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

[31] Iebeling Kaastra and Milton Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215–236, 1996.

[32] E Michael Azoff. *Neural network time series forecasting of financial markets*. John Wiley & Sons, Inc., 1994.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,

M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

[35] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[36] Peter Malec. A semiparametric intraday garch model. *Browser Download This Paper*, 2016.

[37] Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy. Hawkes processes in finance. *Market Microstructure and Liquidity*, 1(01):1550005, 2015.

[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[39] Rob Reider. Volatility forecasting i: Garch models. *New York*, 2009.

[40] Tohru Ozaki. Maximum likelihood estimation of hawkes' self-exciting point processes. *Annals of the Institute of Statistical Mathematics*, 31(1):145–155, 1979.

[41] Yuanda Chen. Thinning algorithms for simulating point processes. 2016.

[42] Szabolcs Mike and J Doyne Farmer. An empirical behavioral model of liquidity and volatility. *Journal of Economic Dynamics and Control*, 32(1):200–234, 2008.

[43] Dan Amiram, Balazs Cserna, and Ariel Levy. Volatility, liquidity, and liquidity risk. 2016.

[44] Dave R Gargett. The link between stock prices and liquidity. *Financial Analysts Journal*, 34(1):50–54, 1978.