Reinforcement Learning for Statistical Arbitrage

Project 4: STA2536:Data Science for Risk Modelling

Authors:
Boris Migan: 1009644725
John Ndolo: 1009482560
Michael Luan: 1004282867

Master of Financial Insurance

DEPARTMENT OF STATISTICAL SCIENCES,
FACULTY OF ARTS & SCIENCE

January 16, 2023

Abstract

In this project, we develop an optimal trading strategy that aims to maximise the profit of a trader. We develop a double deep Q-learning approach to optimize a trading strategy for a financial asset with price evolution described by an Ornstein-Uhlenbeck process. The trader's actions, which represent changes in inventory position, is also able to affect the asset's price evolution at every time step. The reward for each time step is based on the change in the book value of the shares held and the cost of changing the inventory position. The goal is to maximize the expected sum of rewards by the end of the time horizon, while ensuring that the final inventory is zero. The algorithm is trained using simulated data and the results presented in terms of the optimal strategy over time and how it varies based on time, price, and inventory. We also investigate the role of various parameters in the model (κ, β, and η) by seeing its effect on the final sum of rewards.

Contents

List of Figures

1 Introduction

In the field of financial trading, it is crucial to make well-informed decisions in order to achieve the highest profits and minimize potential losses. One way to accomplish this is by utilizing machine learning algorithms to analyze past data and forecast future market movements, but how good is too good? In this particular project, we utilize a machine learning technique called reinforcement learning to develop an optimal trading strategy for a financial asset whose price is modeled using a discrete version of the Ornstein-Uhlenbeck process. The Ornstein-Uhlenbeck process is a continuous-time stochastic process that is frequently used to model financial time series, such as stock and bond prices. It is characterized by a mean-reverting behavior, meaning that the asset's price tends to gravitate towards a long-term average value. To create a discrete version of this process, we approximate the continuous-time version using discrete time steps. The resulting discrete Ornstein-Uhlenbeck process can then be used to describe the evolution of an asset's price over time.

Reinforcement learning is a type of machine learning in which an agent learns to interact with its environment in order to maximize a reward [1]. In the context of financial trading, the agent is the trader and the environment is the market. The trader takes actions, such as buying or selling shares, and receives a reward based on the outcome of those actions. By learning from past experiences, the trader can adapt its strategy and make better decisions in the future. One specific type of reinforcement learning that we use in this project is called double deep Q-learning. This algorithm uses two neural networks to approximate the Q-function, which represents the expected reward for a given state and action. The first network is used to select the best action, while the second network is used to evaluate the action chosen by the first network. This approach helps to address the problem of overestimation, which can occur when using a single network to approximate the Q-function.

In this project, the trader's actions are represented by changes in inventory position, and the reward is based on the change in the book value of the shares held and the cost of changing the inventory position. The goal is to maximize the expected sum of rewards by the end of the time horizon, while ensuring that the final inventory is zero. The Algorithm is trained using 10,000 paths of simulated data and the results visualized in terms of the optimal strategy over time and how it varies with respect to the trader's inventory position and the asset's price. We will also investigate the effect of the various parameters (k, β and η) on the performance of the algorithm. Overall, this project demonstrate the potential of reinforcement learning for financial trading and provide insights into the factors that influence trading performance (what is in modern finance referred as algorithmic trading).

The structure of our project takes a form of four parts which include: the introduction which gives overview on the objectives of the project, the methodology which explains the approaches taken to arrive at the desired objectives, the results and discussion part which explains the outputs of the algorithms and finally the conclusion which gives a summary of our observations and findings.

2 Methodology and Model Setup

In this section, we present the concepts and methods we utilize to obtain our results. We first introduce the case study and the Ornstein-Uhlenbeck (OU) process, which serves as the foundation for modeling the collection of stock prices, denoted as S, as a discrete process. Next, we delve into the machine learning concepts and techniques that we apply in our implementation.

2.1 Case setup and the Ornstein-Uhlenbeck process (OU)

Generally, the OU process is defined by the following Stochastic Differential Equation (SDE);

$$dS_t = \kappa(\theta - S_t)dt + \sigma dW_t$$

where W_t is a standard Brownian motion, and $\kappa > 0$, θ, and $\sigma > 0$ are constants. The analytical solution of the SDE is derived as follows, first we know that θ is the long-term mean of the process S_t thus we can simplify the SDE by introducing the change of variable

$$Y_t = S_t - \theta$$

that subtracts off the mean. Then Y_t satisfies the SDE:

$$dY_t = dS_t = -\kappa Y_t dt + \sigma dW_t.$$

In the SDE of Y_t, the process Y_t is seen to have a drift towards the value zero, at an exponential rate κ. This motivates the change of variables below,

$$Y_t = e^{-\kappa t} Z_t \quad \Longleftrightarrow \quad Z_t = e^{\kappa t} Y_t,$$

which should remove the drift. A calculation with the product rule for Itô integrals shows that this is so:

$$dZ_t = \kappa e^{\kappa t} Y_t dt + e^{\kappa t} dY_t = \kappa e^{\kappa t} Y_t dt + e^{\kappa t}(-\kappa Y_t dt + \sigma dW_t) = 0dt + \sigma e^{\kappa t} dW_t.$$

The solution for Z_t is immediately obtained by Itô-integrating both sides from s to t:

$$Z_t = Z_s + \sigma \int_s^t e^{\kappa u} dW_u.$$

Reversing the changes of variables, we have:

$$Y_t = e^{-\kappa t} Z_t = e^{-\kappa(t-s)} Y_s + \sigma e^{-\kappa t} \int_s^t e^{\kappa u} dW_u,$$

and

$$S_t = Y_t + \theta = \theta + (S_s - \theta)e^{-\kappa(t-s)} + \sigma \int_s^t e^{-\kappa(t-u)} dW_u.$$

By using the solution of the general OU process above, in this project we suppose that the mid-price of an asset price $(S_t)_{t=0,1,\cdots,T}$ evolves in (discrete) time as follows

$$S_t = \theta + (S_{t-1} - \theta)e^{-\kappa} + \eta Z_t$$

where where $Z_1, \cdots, Z_T$ are assumed to be iid in the context of this project. and $Z_1 \sim N(0,1)$. Hence, S is a discrete version of an Ornstein-Uhlenbeck process.

In addition, in this project, when the trader sends an order to the market, that action modifies the price evolution to the following:

$$S_t = \theta + (S_{t-1} - \theta)e^{-\kappa} + \beta\,\text{sgn}(q_t - q_{t-1})\sqrt{|q_t - q_{t-1}|} + \eta Z_t,$$

The additional term in our equation represents the change in the price of the asset due to the trader's action. We also assume that at the end of the time horizon, the trader liquidates all of its assets.

2.2 Reinforcement Learning

Reinforcement learning involves three key components: the environment, the agent, and the actions that the agent can take within the environment [1]. At each time step $\{t = 1, \cdots, T-1\}$, the agent begins in a state $S_t \in S$, takes action $a_t \in \mathbf{A}$ according to a policy characterized by parameters Θ with $\Pi^{\Theta} : S \longrightarrow \Pr(A)$, where $\Pr(A)$ is the probability distribution on all actions and can be either deterministic or randomized. The agent then transitions to a new state $S_{t+1} \in \mathbf{S}$ based on unknown transition probabilities $\Pr(S_{t+1} = S | S_t = S, a_t = a)$ and receives a reward $r_t = r(S_t, a_t, S_{t+1}) < \infty$. In that regard, in this project, to develop a trading strategy we first define the environment in which the agent, or the trading system, will operate. This environment includes information about prices,inventory and other relevant market conditions (time). The goal of the agent is to make decisions based on this information in order to maximize the reward, which in our implementation is the profit generated by the trades. To determine the reward, we consider two factors: the change in the book value of the shares held by the agent and the cost of changing the inventory position. The reward function we consider is given as,

$$rt = q_t(S_{t+1} - S_t) - [(q_t - q_{t-1})S_t + \Delta|q_t - q_{t-1}|]$$

where, the first term, $q_t(S_{t+1} - S_t)$, represents the change in the book value of the shares that you hold over the period and the second term, $[(q_t - q_{t-1})S_t + \Delta|q_t - q_{t-1}|]$, represents the cost of changing your position from q_{t-1} to q_t.

Generally, the ultimate performance goal of an agent in reinforcement learning is to maximize the total discounted future rewards or the average future reward in the limit of an infinite time horizon generated by a specific sequence of actions taken [2]. This performance goal can be expressed by a value function as:

$$V^{\pi}(S) = \sum_{t=0}^{\infty} \gamma^t R_{\pi}^t | S_0 = S$$

where R_{π}^t denotes the rewards the agent receives when using policy π. The goal of the agent is to find the optimal policy π which maximizes value action equation. The performance goal evaluated under the optimal policy is called the value function, which is denoted as $V(S) := V^{\pi}(S)$. In addition to the value function, we also define a Q-function which contains both the state and the action:

$$Q(S,a) = \mathbb{E}[R_a^t + \gamma V(S_{t+1})|S_t = S]$$

where Q is short for the quality of an action at a given time point in a state space. By definition,

$$V(S) = \max_{a \in A} Q(S,a)$$

. Substituting this back into the definition of the Q-function gives:

$$Q(S,a) = \mathbb{E}[R_a^t + \gamma \max_{a' \in A} Q(S_{t+1}, a')|S_t = S].$$

This is known as the Bellman equation.

2.3 Q-learning

Generally, in Q-learning, the goal is to learn the optimal action-value function, denoted as $Q^*(s,a)$, which is the maximum expected future reward for taking action a in state s and following the optimal policy thereafter. The optimal action-value function can be expressed using the Bellman equation (It is a recursive relationship that defines the optimal action-value function in terms of itself).

$$Q(S,a) = \mathbb{E}[R_a^t + \gamma \max_{a' \in A} Q(S_{t+1}, a')|S_t = S],$$

or

$$Q(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} Q(s',a')]$$

where $p(s',r|s,a)$ is the probability of transitioning to state s' and receiving reward r given that action a is taken in state s, γ is the discount factor, and $\max_{a'} Q^*(s',a')$ is the maximum expected future reward for any action a' taken in state s'.

In practice, we cannot directly compute the optimal action-value function because the true transition probabilities and rewards are unknown [3]. Instead, we approximate the optimal action-value function using a neural network. This is inspired by the universal approximation theorem which states that a feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to an arbitrary degree of accuracy, provided that the activation function used is continuous. Therefore we can approximate the optimal value function under this assumption. By definition, the mathematical formula for the universal approximation theorem is:

$$\forall f(x) \in C(\mathbb{R}^n)$$

, $\exists$ a feedforward neural network $G(x,\mathbf{w})$ with a single hidden layer and an activation function $\sigma(x)$ that is continuous and non-constant, such that:

$$\|f(x) - G(x,\mathbf{w})\|_\infty \leq \epsilon$$

where $\mathbf{w}$ represents the weights and biases of the network, and ϵ is an arbitrarily small positive constant. Therefore, by this theorem, at each iteration of the neaural network, we can select an action according to an ϵ-greedy policy and update the action-value function using the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right)$$

where α is the learning rate, γ is the discount factor, and $Q(s_t, a_t)$ is the current estimate of the action-value function for taking action a_t in state s_t. The term $r_t + \gamma \max_a Q(s_{t+1}, a)$ is the target value for the update, which represents the expected return for taking action a_t in state s_t. The current estimate of the action-value function is then updated towards the target value by a factor of α. The Q-learning method updates the estimates for the action-value function for a given state and action in the reinforcement learning problem under study, where in this case is optimise the profit at each point in time. Based on the Bellman equation, which states that the value of a state-action pair is equal to the expected sum of future rewards, discounted by a factor of gamma at each time step, our problem under study can be simply represented by this Bellman principle expression at each point in time;

$$Q(S, I, T, a) = r + \gamma \max_{a'} Q(S', I', T', a')$$

where $Q(S, I, T, a)$ is the expected discounted sum of rewards from taking action a at state (S, I, T), r is the immediate reward received from taking action a at state (S, I, T), γ is the discount factor, and the $\max_{a'} Q(S', I', T', a')$ term represents the maximum expected discounted sum of rewards from all possible actions at the next state (S', I', T'). However, one drawback of Q-learning is that it may exhibit an overestimation bias, leading to overestimation of action-values. This can result in sub-optimal policies and poor performance in certain situations. To address this issue, several variants of Q-learning have been proposed, such as double Q-learning that we introduce in the following subsection.

2.4 Double deep Q-learning

Double Q-learning is a variant of Q-learning that helps to mitigate the over-estimation bias that can occur when using Q-learning. In double Q-learning, we maintain two separate action-value functions, $Q_1(s, a)$ and $Q_2(s, a)$, and at each iteration we select the action according to an ϵ-greedy policy using the following rule:

$$a = \begin{cases} \arg\max_{a'} Q_1(s, a') & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

Where ϵ is a probability that determines the balance between exploration and exploitation and is usually chosen to be small. Then, we update the action-value functions using the following rule:

$$Q_1(s, a) \leftarrow Q_1(s, a) + \alpha[r + \gamma Q_2(s', \arg\max_{a'} Q_1(s', a')) - Q_1(s, a)]$$

$$Q_2(s, a) \leftarrow Q_2(s, a) + \alpha[r + \gamma Q_1(s', \arg\max_{a'} Q_2(s', a')) - Q_2(s, a)]$$

where $\arg\max_{a'} Q_1(s', a')$ and $\arg\max_{a'} Q_2(s', a')$ are the actions that maximize the action-value functions Q_1 and Q_2, respectively, in the next state s'. By using two separate action-value functions to derive the double deep Q-learning equation, we begin with the Bellman equation for Q-learning:

$$Q^*(s,a) = r + \gamma \max_{a'} Q(s',a')$$

where $Q^*(s,a)$ is the optimal action-value function, r is the reward for taking action a in state s, γ is the discount factor, and s' is the next state. The equation states that the optimal action-value is equal to the immediate reward plus the discounted maximum action-value of the next state.
To apply this equation to our problem, we estimate the action-value function $Q(s,a)$ using a neural network. We do this by minimizing the mean squared error between the predicted action-values and the target action-values:

$$L = \frac{1}{N}\sum_{i=1}^{N}\left(Q(s_i,a_i) - \left(r_i + \gamma \max_{a'} Q(s_i',a')\right)\right)^2$$

Where N is the number of samples in the training batch and s_i, a_i, r_i, and s_i' are the state, action, reward, and next state for sample i, respectively. To improve the stability of training, we use a separate network to estimate the target action-values rather than using the same network. This is the key idea behind double deep Q-learning. The target network is a copy of the main network and is updated less frequently, typically by copying the weights from the main network every few iterations. The double deep Q-learning equation is then:

$$L = \frac{1}{N}\sum_{i=1}^{N}\left(Q(s_i,a_i) - \left(r_i + \gamma \max_{a'} Q_{\text{target}}(s_i',a')\right)\right)^2$$

Where $Q_{\text{target}}(s,a)$ is the action-value estimate of the target network. This equation is implemented in PyTorch by defining a loss function using the mean squared error criterion, passing in the predicted and target action-values, and then calling the backward() method on the loss to compute the gradients and update the weights of the main network using an optimizer.

To generalise the idea of Deep Q-learning as an extension of Q-learning that uses a neural network to approximate the action-value function, we have the goal as to learn the optimal parameters (weights and biases) of the neural network, denoted as θ, such that the action-value function $Q(s,a;\theta)$ is as close as possible to the true action-value function $Q^*(s,a)$. To estimate The optimal parameters, the following loss function is used;

$$L(\theta) = \mathbb{E}_{s,a,r,s'}\left[(y - Q(s,a;\theta))^2\right]$$

where $y = r + \gamma \max_{a'} Q(s',a';\theta)$ is the target value and the expectation is taken over a batch of transitions (s,a,r,s') experienced by the agent. The loss function measures the mean squared error between the target value and the predicted action-value function. To optimize the loss function, we use stochastic gradient descent. Given a set of transitions (s,a,r,s'), we compute the gradient of the loss function with respect to the parameters as follows:

$$\nabla_\theta L(\theta) = \mathbb{E}s,a,r,s'\left[(y - Q(s,a;\theta))\nabla\theta Q(s,a;\theta)\right]$$

We then update the parameters using the gradient:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L(\theta)$$

where α is the learning rate.

2.5 Neural networks

A neural network is a type of machine learning model that is designed to mimic the functioning of the human brain. The architecture of a neural network consists of the following components: an input layer, one or more hidden layers, and an output layer. The input layer receives the input data and passes it on to the hidden layers. The hidden layers process the data and pass it on to the output layer, which produces the output. Each layer consists of a set of interconnected nodes or neurons. The connections between the nodes are weighted, and each node applies a nonlinear function (such as a sigmoid function) to the weighted sum of its inputs. The figure below illustrates a simple feed forward network as a representation of general architecture of a neural network;

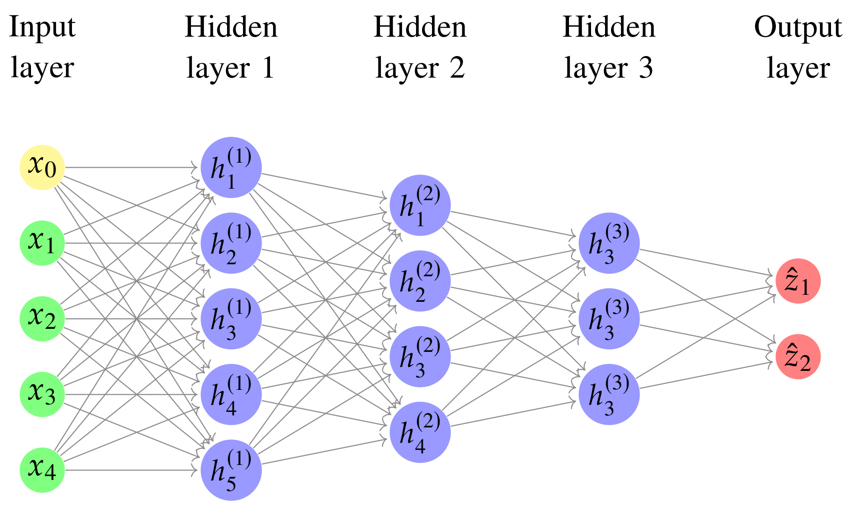

Figure 1: A three layer neural network.

The weights and biases of the network are adjusted during training, using an optimization algorithm such as stochastic gradient descent. The goal of training is to find a set of weights and biases that minimizes the error between the predicted output and the true output.
In the context of our project, the neural network is used to approximate the Q-function, which represents the expected future rewards for each action in a given state. The input to the network is the current state (asset price and inventory), and the output is the Q-values for each action (short, nothing, long). The Q-values are updated during training using the Bellman equation, with iterations until convergence. Mathematically, the Q-learning equation can be written as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

where s and s' are the current and next states, a and a' are the current and next actions, r is the reward, γ is the discount factor, and α is the learning rate (For our purposes, we set discount factor to 1 and we set learning rate to 0.001). In the case of double deep Q-learning, the Q-values are updated using the Q-values from both the main and target networks. This helps to stabilize the learning process and improve the accuracy of the Q-values. The double deep Q-learning equation can be written as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma Q_{target}(s', \arg\max_{a'} Q_{main}(s',a')) - Q(s,a) \right)$$

where Q_{main} and Q_{target} are the Q-values from the main and target networks, respectively. The $\arg\max$ function returns the action with the maximum Q-value.

3 Results and discussions

In the following results and discussion section, we present the findings of our analysis of the double deep Q-learning algorithm applied to the problem of optimal stock trading. We first provide an overview of the performance of the algorithm and then delve into a detailed analysis of the effects of various parameters on the solution. Additionally, we explore the behavior of the optimal strategy over time and how it varies with changes in price and inventory. Finally, we will conclude with a summary of our main findings and discuss potential directions for future work. The results of our discussion are obtained by considering the impact of various parameters, including

$$S_0 = 1, \theta = 1, \kappa = 0.5, \eta = 0.02, \beta = 0.01, Q = 10, T = 100$$

on the evolution of stock prices and the resulting inventory positions and single-period rewards for trading. For instance, with respect to the problem at hand, the figure below shows a single sample path (in blue), the confidence bands (in dashed black), some sample paths (light multi colored), for a strategy that sells the asset from time 0 to time 5 and then stops trading all together.

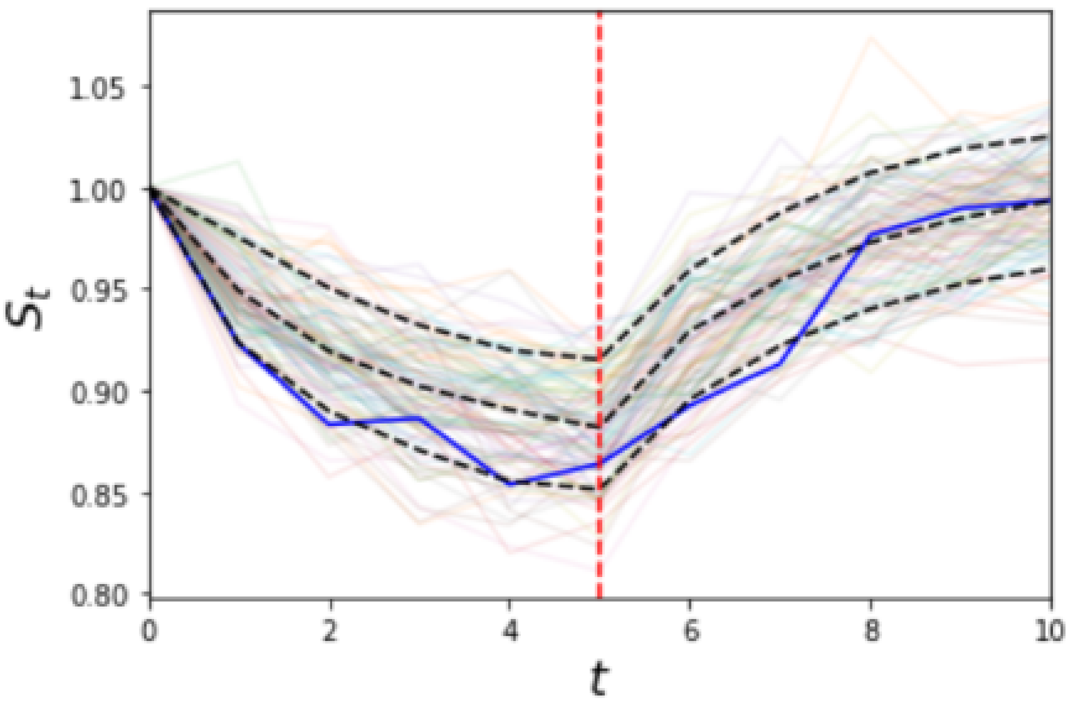

Figure 2: Single sample path of assets trading strategy

We notice that the price is initially pushed down, despite starting at the mean-reversion level, and then returns back to this level after trading has ceased. Based on these observations, we delve further into the results and present them in the following subsections, where we elaborate on the various ways in which the optimal strategy could be influenced by time, price, and inventory, as well as investigating the role played by the parameters κ, β, and η in shaping these variations.

3.1 Visualizing the Optimal Strategy Through Time

The figures below illustrate the results before training our model with an initial random action. Figure (3) with four subplots, arranged from left to right and top to bottom, depict the evolution of the stock price, the accumulated expected reward, the inventory, and the actions taken as a function of time. The top left plot displays the evolution of the stock price, including a dotted line representing the long-term mean reversion rate.

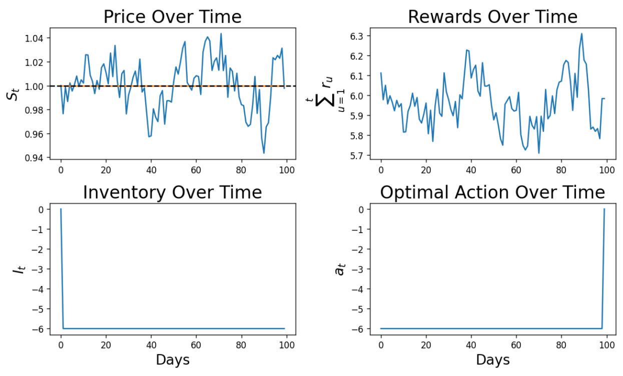

Figure 3: Optimal strategy through time at initial iteration

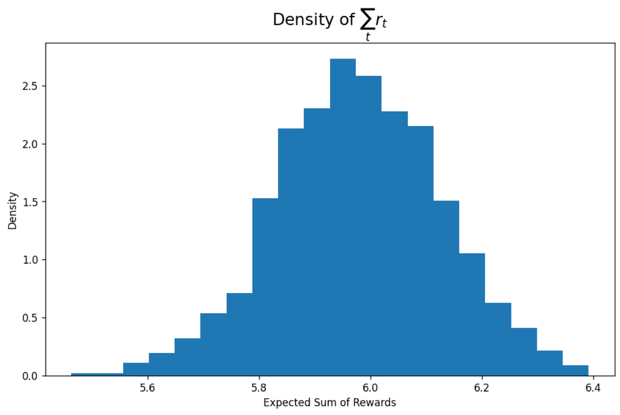

Figure 4: Histogram of total rewards at initial iteration

It can be observed that the inventory drastically decreases from the beginning then stagnates, indicating that the initial random action involves shorting the asset which decreases the inventory. As time progresses, the action remains the same, which is reflected in the sum of the rewards, which exhibits an negative relationship with the stock price and can be further explained by looking at the action plot on bottom right (opposite) of the inventory plot, we see that the action involves selling one unit of the stock. This suggests that at the beginning of the training, the model has not yet learned an optimal strategy and it is not yet clear whether the strategy is effective or not. The density plot of the expected sum of rewards in Figure (4) shows a bell-shaped histogram, indicating that the distribution is approximately symmetric. The mean of the distribution appears to be around 1, with the majority of the prices concentrated within the range of 0.9 to 1.1. The plot also appears to be slightly skewed to the right, with a longer tail extending towards higher values of the expected sum of rewards. The expected sum of rewards is likely to fall within a certain range of values, with the highest probability of occurrence around the mean of 1, as shown by the shape of the curve in the density plot. This becomes more evident as the number of iterations increases. The presence of a Gaussian distribution may suggests that the expected sum of reward is likely to fall within a certain range of values, with the highest probability of occurrence concentrated around the mean of about 1 on this plot.

The figures below depict the results of the model after 2000 iterations. Figure (5) illustrates the temporal evolution of the stock price, the accumulated expected reward, the inventory position, and the actions taken at each time point. The density of the expected rewards is shown in Figure (6), and the loss function is depicted in Figure (7). From Figure (7), the loss function shows a clear trend of decreasing over time. This suggests that the model is continuously learning and improving its performance as it processes more data. This improvement is also reflected in the slight learning progress observed in the optimal strategy through time plotted in Figure (5).

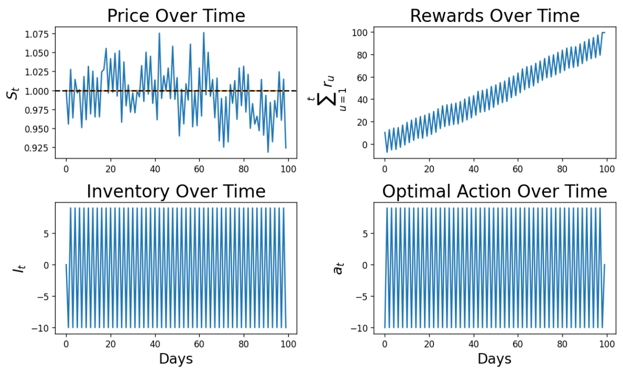

Figure 5: Optimal strategy through time

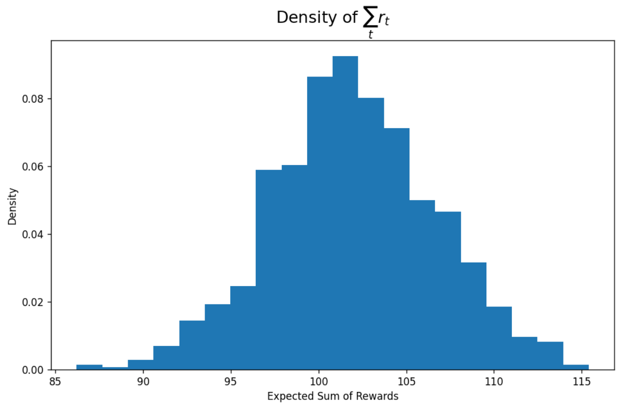

Figure 6: Histogram of total rewards

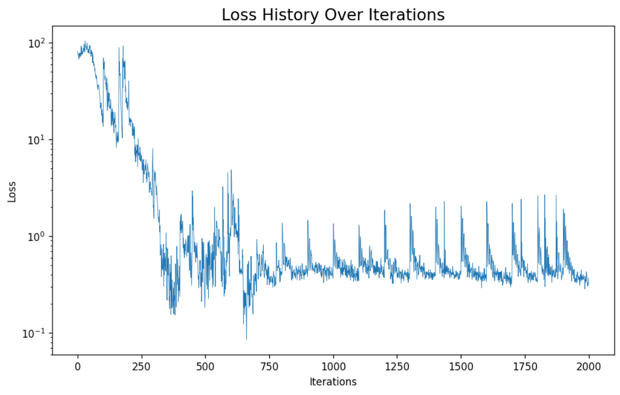

Figure 7: Loss function

just like in Figure (3),the inventory plot in Figure (5) and the entire model's learning process show that the model initially engaged in a short position and then learned for a short time and stagnated for the rest of training period. This strategy resulted in a sum of expected rewards that fluctuated around a mean value of 100, as seen in the histogram of the density of the expected rewards in Figure (6). The distribution of these rewards appears to be somewhat Gaussian, with a range between 90 and 115, however the histogram is slightly skewed to the left. These findings suggest that the model was slightly take the right course of determining the optimal trading strategy.

The figures below show the final results of the 4000th iteration. The Figure (10) depicts the history of the loss over iterations. It can be observed that after a significant decrease from 0 to 2000 iterations, the loss seems to converge slightly above 10^{-1}. However, it is worth noting that the loss appears to converge but oscillate above the minimum loss achieved thus far, which is approximately 10^{-3}. There may be several reasons for this phenomenon, including local minima, an inappropriate learning rate, an inadequate optimization algorithm, a non-convex loss function, or noise in the data. In Figure (8), we observe a clear upward trend in the sum of expected reward as time increases this pattern could be described as an "ascending sawtooth pattern." or "ascending peaks and troughs pattern." It may be due to dynamic market conditions. Due to the constant change in the market conditions, which may cause the model's performance to fluctuate.

On the other hand the plot of both the inventory and actions illustrating a "sawtooth pattern" as well.

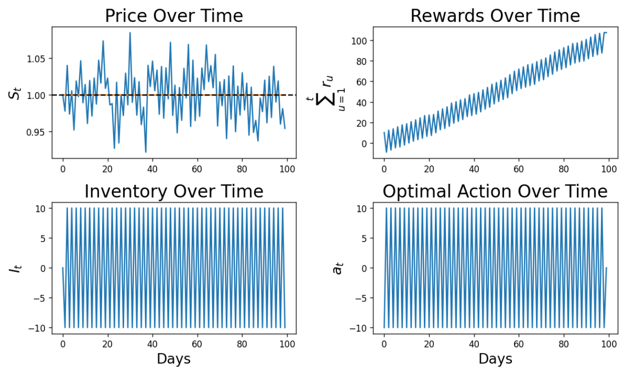

Figure 8: Optimal strategy through time

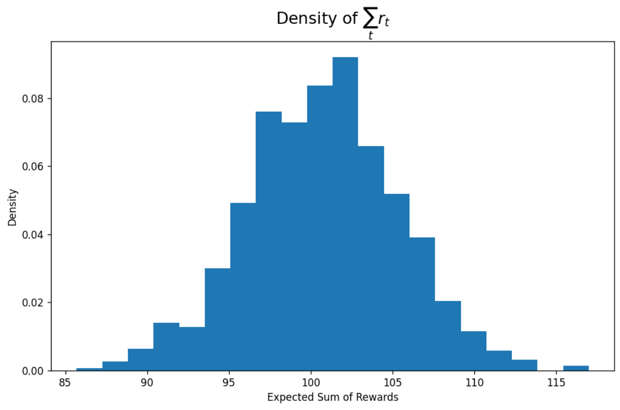

Figure 9: Histogram of total rewards

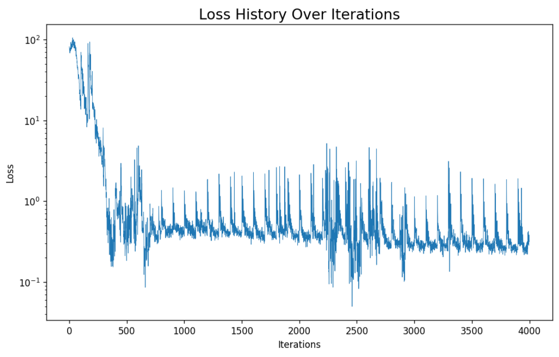

Figure 10: Loss function

In addition, in Figure (9), the plot of the density of the sum of expected reward after 4000 iterations has an even wider range bounded by 80 to 120 with a mean of about 103. The density looks more gaussian like and symmetric compared to the previous density function.

In summary, the model seems to have learned an optimal trading strategy that resulted in an increasing sum of expected rewards through time. The distribution of these rewards appears to be somewhat Gaussian, with a range between 80 and 120. The model's behavior can be attributed to market dynamics, as it follows the laws of demand and supply to make profitable trades. The patterns observed in the inventory position and expected rewards suggest that the model has successfully learned to maximize profit by learning it can influence the price of the asset the next next day by taking the best option which happens to be most extreme action.

3.2 Visualizing the Optimal Strategy as a Function of Time, Price, & Inventory

The plot below demonstrates that the optimal trading strategy of a handful of simulations that varies with respect to time, inventory, and stock price. The optimal strategy involves a combination of both selling and buying actions, color-coded in red, blue, and black respectively to represent the strategies: hold 10, hold -10, and hold zero of the asset. We see at the first time step, the agent decides that the optimal decision is always to be at -10 held assets. We see for the final time step, the agent liquidates all the assets as is described. The model appears to takes the maximum action at each step, as it has learned that trading actions can influence the asset price and maximizing these actions can lead to greater control of the price process ultimately leading to higher profits. This implies that the strategy seems to be invariant of both price and time and seems to only be influenced by the inventory. This is supported by Figure (8) where we see that the optimal action over time oscillates from -10 and 10 until the final time step where the trader is forced to liquidate the inventory. The inventory follows suit through time where it is always one step behind the optimal action. Due to this, we see that the price of the asset over time is heavily impacted by the trader's actions as when we compare the asset price over time from Figure (8) to that of in Figure (3), the price exhibits much more oscillations. Consequently we also see that the rewards over time slowly but surely rises linearly as the agent has "learned" its influence on the asset price.

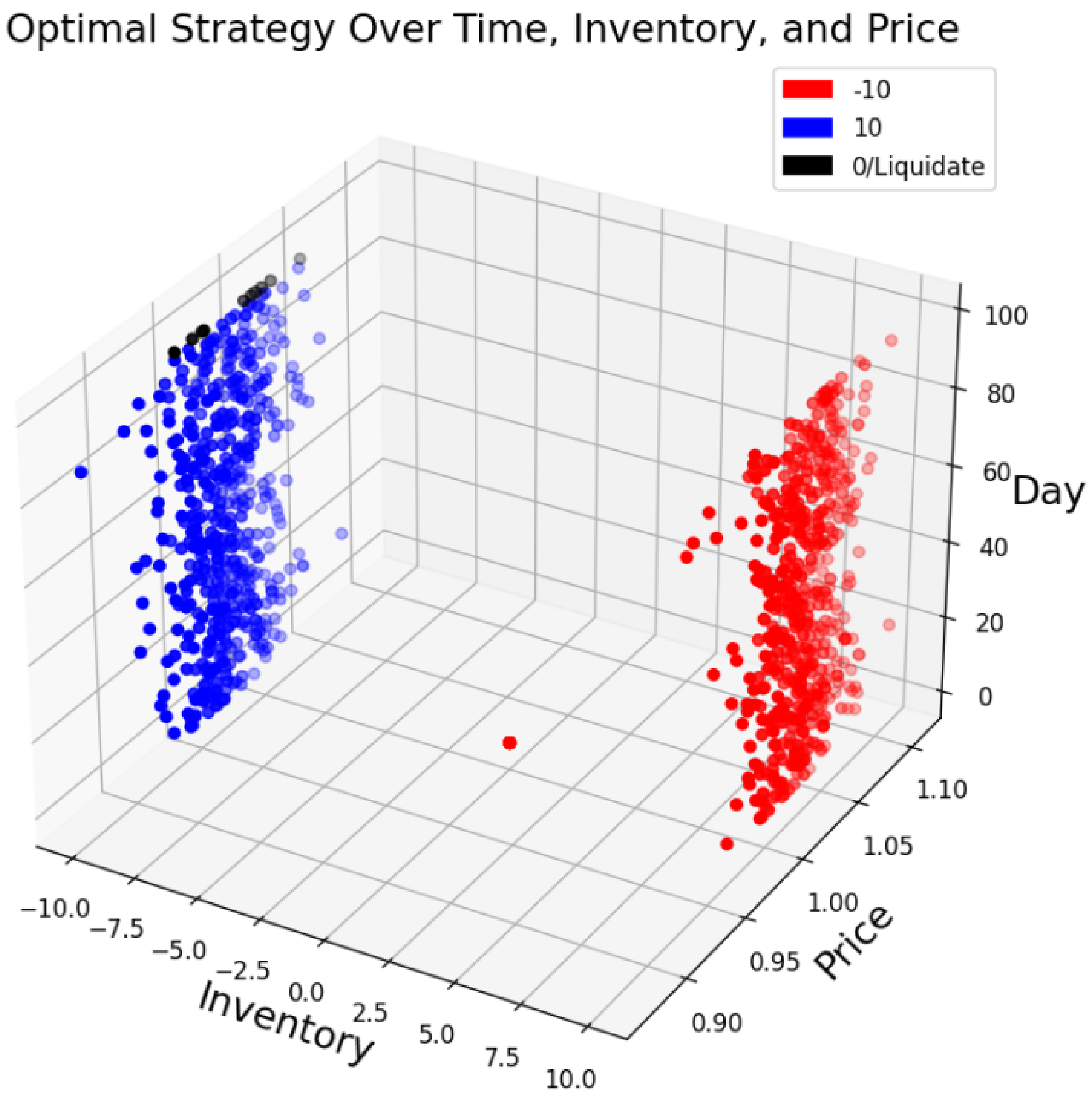

Figure 11: Optimal Strategy as a Function of Time, Price, & Inventory

3.3 Investigating the Role of κ, β, and η

By definition, κ is the mean-reversion rate of the stock price process, it determines how quickly the stock price will return to its long-term mean value. A higher value of κ means that the stock price will return to its mean value at a faster pace, which could affect the optimal trading strategy as we can see on the figure below. Normally, mean reversion demonstrates a form of symmetry about θ since a stock may be above its historical average approximately as often as below. This is evident in the Figure (12) the *symmetry* of the density plot of the expected sum rewards improves with the increase in the κ value.

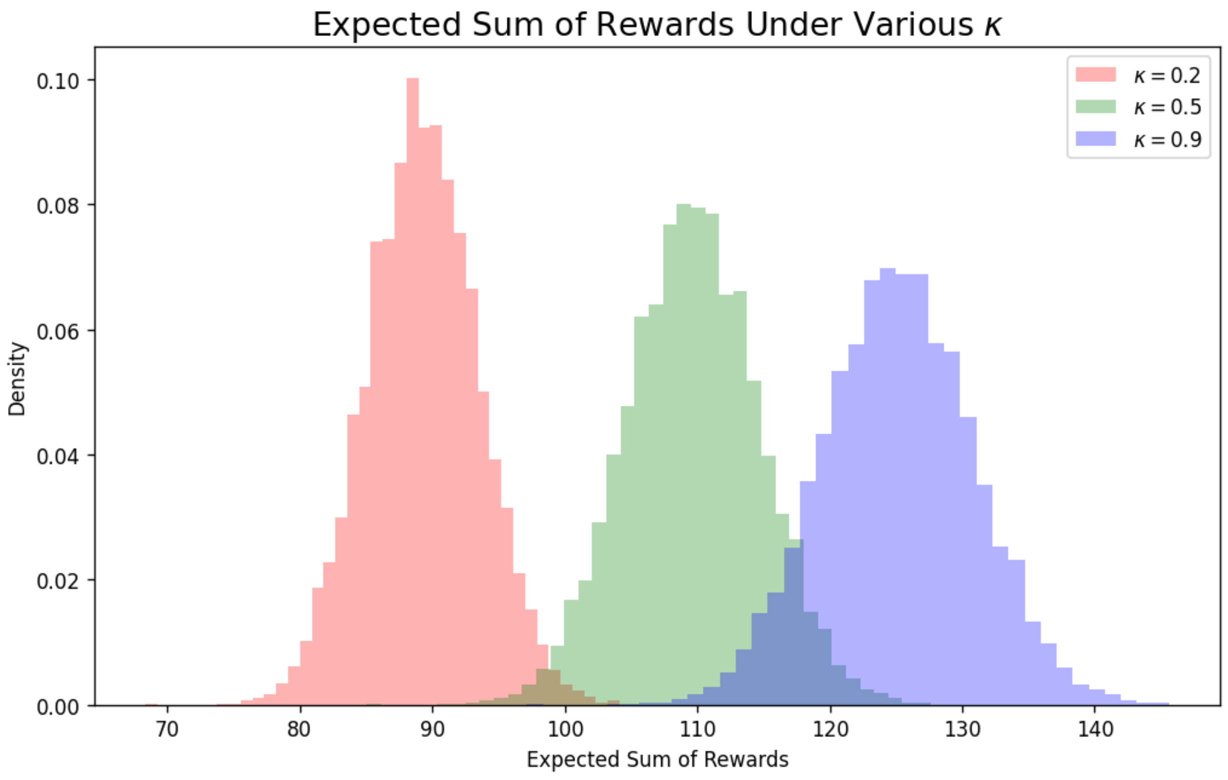

Figure 12: The role of $\varkappa$

Intuitively, a higher value of κ may result in a trading strategy that places a greater emphasis on short-term fluctuations in the stock price, potentially involving more frequent trades to take advantage of these movements along with its strategy of market manipulation as market correction compensates better for the trading impact which allows the trader to make more greater profits over time compared to a slower market correction rate which may not accommodate the agent's aggressive strategy. Additionally, a smaller value of κ may correspond to a strategy that incorporates long-term trends and potentially involves holding positions for longer periods of time. In the figure above we observe that as the value of κ increases the sum of reward which in lieu increases with the value of κ. This increase demonstrated when κ is 0.2, 0.5, and 0.9. The approximate mean of the expected sum of rewards for the respectively κ values is approximately around 88, 108 and, 126. Moreover, we see in the figure above that the density curve becomes wider and less peaked as the value of κ increases. This suggests that the expected rewards increase with an increase in the value of κ, resulting in a better trading strategy.

Increasing the value of β can have a significant impact on the price evolution, as it represents the severity of the influence that an action such as buying or selling has on the price. This can potentially create opportunities for arbitrage, and as seen in the figure below, it can result in higher rewards when the effects of these changes are studied. From the figure below, we can clearly see that there is a positive correlation between the value of β and the mean of the expected sum of rewards. The density of the expected sum rewards widens with increase in the β value. We also note that, for values of β equal to 0.005, 0.01, and 0.015, the mean of the expected sum of rewards is approximately 17, 74, and 164 respectively.

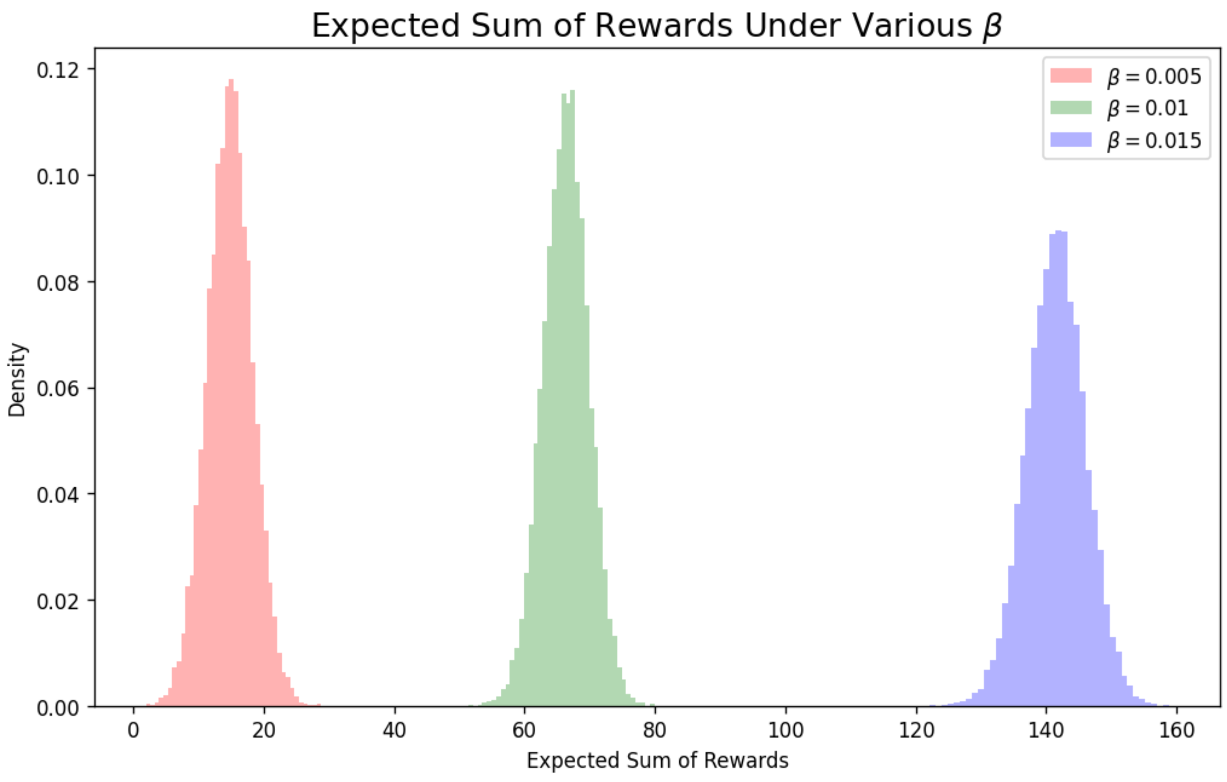

Figure 13: The role of β

These observations are arguably due to the increased profit opportunity introduced by increasing the significance of the impact from the trader's actions on the stock price.

The ηZ_t term represents the random fluctuations in the stock price introduced by external factors such as market sentiment and other unpredictable events. It could be described as the "volatility". η is a coefficient indicating the severity of said volatility, i.e. as η increases the stock price becomes more volatile and vise-versa.

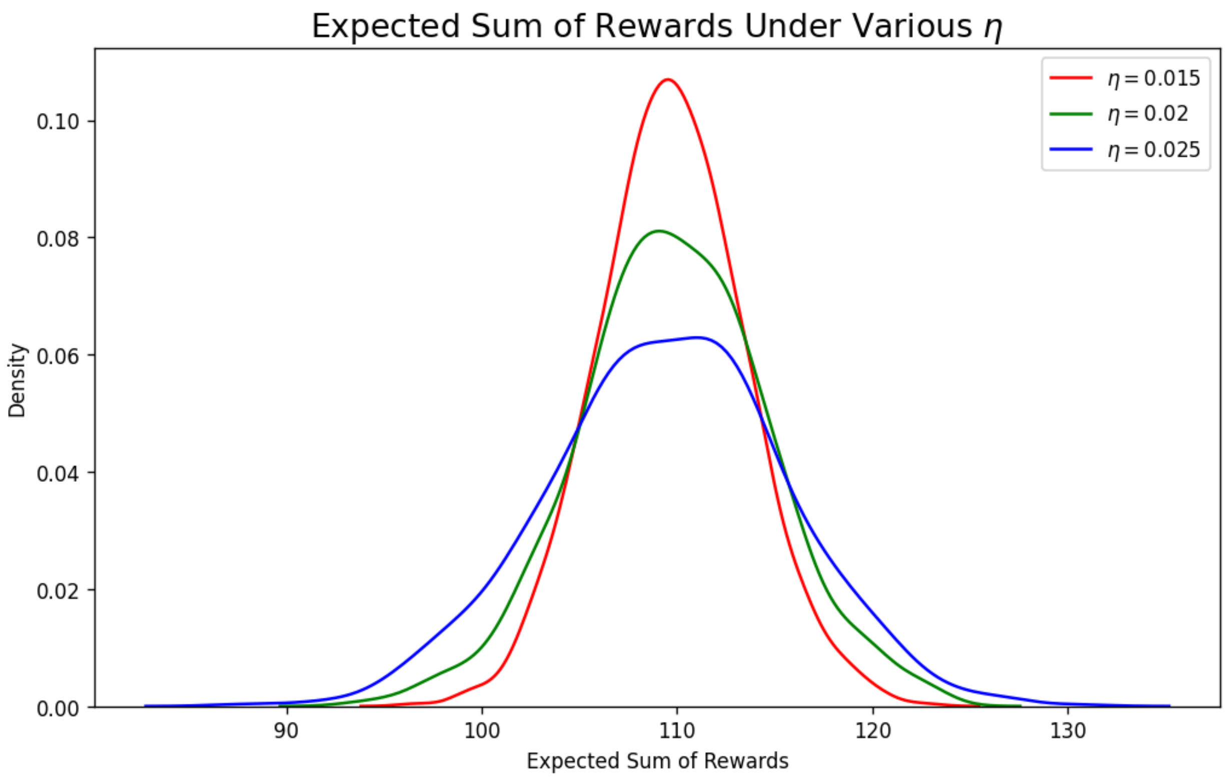

Figure 14: The role of η

The plot above illustrates the relationship between the parameter η and the density of the expected sum of rewards. As the value of η increases the density seams to become flatter, resulting in a wider spread of the expected sum of reward. For value of $\eta = 0.015, 0.02, 0.025$ the corresponding interval of expected sum of rewards are approximately (20,135), (90,128) and (110,125) respectively. Intuitively this trend makes sense as with higher volatility traders are more often exposed to higher gains or higher losses contributing to greater weight in the tails of the distribution..

4 Conclusion

In conclusion, our investigation into the optimal trading strategy using double deep Q-learning has yielded several key findings. Firstly, we found that the parameters κ, β, and η play a crucial role in the performance of the strategy. By varying these parameters, we were able to visualize how the optimal strategy changes over time, price, and inventory. This allowed us to develop an approach for visualizing the optimal strategy as a function of these variables.

We found that increasing the mean reversion rate parameter κ compliments a more aggressive trading strategy. The impact that parameter β exhibits was found to have a positive effect on the expected sum of rewards with larger values of β resulting in higher expected returns. Finally, the volatility parameter η was found to have a "spread out" effect on the expected sum of rewards, with larger values of η resulting possibly higher but also lower expected returns.

Furthermore, we made plots of the optimal strategy through time, and observed how it evolves as the price and inventory change. We found that the strategy is highly dynamic, and is able to adapt to changing market conditions in order to maximize profits by learning to exploit its effect on the asset price through the trading impact. This is a key advantage of using double deep Q learning, as it allows the strategy to learn and adapt over iterations of training. The action of optimal trading strategy is invariant of time and asset price. It was found that the strategy varies only with respect to inventory. The model incorporates market dynamics such as selling high and when inventory is high and buying low and when inventory is low.

Overall, our analysis has provided valuable insights into the use of double deep Q-learning for optimal trading by carefully considering the parameters and visualizing the results. We have been able to develop a competitive strategy that is able to adapt to changing market conditions and maximize profits.

References

[1] C. Alvaro and S. Jaimungal, *Machine Learning and Algorithmic Trading*. Unpublished Manual, 2023.

[2] B. Ning, F. H. T. Lin, and S. Jaimungal, “Double deep q-learning for optimal execution,” *arXiv preprint arXiv:1812.06600*, 2018.

[3] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.