# thesis

Marko Vukovic

2023-07-03

In this thesis, we aim to address the problem of developing a market making strategy that balances the goal of maximizing returns through spread trading while minimizing risk, particularly inventory risk. This involves training a neural network to act as a market making agent that can adapt to real market conditions through a data-driven approach. The proposed solution must consider constraints such as short selling restrictions and a minimum total value to avoid bankruptcy. The ultimate objective is to create a market making strategy that can effectively manage risk while achieving optimal returns.

## Table of contents

# 1 Introduction

## 1.1 Motivation

The field of deep reinforcement learning (DRL) has been experiencing a rapid growth in recent years, with a plethora of applications across diverse domains including robotics, gaming, and finance. In finance, market making is a crucial activity that involves balancing the supply and demand of securities to ensure their liquidity and stability. However, market making is a complex task that involves managing multiple constraints, such as risk management, regulatory requirements, and market conditions, to name a few.

To address the challenge of market making, DRL has emerged as a promising approach, enabling the creation of autonomous agents that can adapt to changing market conditions and make decisions based on rewards and penalties. However, the conventional DRL models lack the capability of effectively incorporating constraints, resulting in suboptimal or even undesirable outcomes.

The main aim of this thesis is to investigate the challenge of constraint handling in DRL-based market making. The thesis aims to overcome this challenge by incorporating constraints into the DRL model and developing novel algorithms that can optimize the market making strategies while taking into account these constraints. The proposed methods aim to improve the efficiency and robustness of the market making process, enabling the agent to operate within the constraints while maximizing its rewards. The ultimate goal of this research is to contribute to the advancement of DRL in market making and provide a framework for constraint-aware DRL-based market making.

## 1.2 Related work

Previously, reinforcement learning has been applied to the task of market making. In a study by Spooner et al. (2018), a market simulator was developed to generate data for simulations based on historical data, and temporal difference methods were used along with other contributions to increase their effectiveness. This work was expanded upon by Spooner (2020), who created a similar market simulator and trained two agents simultaneously using the SARSA algorithm. One agent served as the market maker, while the other was a new adversarial agent that represented all other market participants, receiving a reward based on the losses of the market making agent. The market making agent trained in this manner was found to be more robust to adverse conditions. Other studies, such as Selser et al. (2021), have also applied deep reinforcement learning algorithms, specifically Q-learning, to the market making problem using their own market data simulator.

To enhance the robustness of the market making agent, the field of safe or constrained reinforcement learning is explored. In Kou et al. (2020), a safety layer was added to the final prediction layer of the agent's neural network to ensure that the actions taken by the reinforcement learning agent do not exceed safety limits. Another approach, as described in Chow et al. (2018), is to use conditional value-at-risk as a percentile risk constraint instead of strict operational limits mapped to the action space. This is particularly relevant for applications in finance, as percentile metrics such as value at risk or conditional value at risk are commonly used for risk management.

## 1.3 Problem statement

In this thesis, we aim to address the problem of developing a market making strategy that balances the goal of maximizing returns through spread trading while minimizing risk, particularly inventory risk. This involves training a neural network to act as a market making agent that can adapt to real market conditions through a data-driven approach. The proposed solution must consider constraints such as short selling restrictions and a minimum total value to avoid bankruptcy. The ultimate objective is to create a market making strategy that can effectively manage risk while achieving optimal returns.

### 1.4 Contribution

## 2 Background

### 2.1 Limit order book

The limit order book is an important component of financial markets as it allows market participants to trade assets at the best available price, ensuring an efficient and fair market. For example, a trader looking to buy an asset can find the best available price by reviewing the sell limit orders in the order book and choosing the lowest price. This helps to ensure that all market participants have equal access to the best available prices for an asset, promoting a fair and transparent market.

Each limit order (LO) must be placed at a specific price level that has been determined by the exchange and is known as the tick size. The tick size is a fixed amount, and the minimum time resolution of exchange market data is known as the tick level.

When a limit order is placed, it can be fulfilled either fully or partially, depending on the volume of the order and the existing orders in the LOB. Orders placed at the current best price are known as market orders, and they are executed immediately as they take the current price. Market orders are different from limit orders as they do not have a specified price level and instead, they take the current price at the time of the execution.

Limit orders are valid until they are cancelled or until their expiry date, which is usually set to 90 days from the date they were placed. When someone refers to the price of a stock, they usually mean the value at the midpoint between the prices for the highest bid and the lowest ask, also known as the market spread. For example, if the highest buy order is at the price level of \$9.5 and the lowest sell order is at the price level of \$10, then the market spread is \$0.5.

When a limit order is received by the exchange, it joins a queue at its specified price level, and it will be matched with an opposite order when all earlier orders at that price level are fulfilled, cancelled, or expire. If a limit order is submitted at a price level that has a matching order in the opposite direction, it will be instantly matched and filled by the exchange. For instance, if a buy order is placed for 100 shares at \$10, and there are two sell orders at \$10 for 50 and 75 shares, respectively, then the first sell order of 50 shares will be completely filled, and the second sell order will be partially filled for 50 out of the 75 shares, and the remaining 25 shares will remain in the LOB with first priority against later orders.

A limit order book is a digital record of all outstanding limit orders for a particular financial asset. A limit order is a type of order placed by a trader that specifies the maximum price they are willing to pay (for a buy order) or the minimum price they are willing to receive (for a sell order) for a certain quantity of a financial asset. The limit order book collects all of these limit orders and displays them in a public and transparent manner, providing an overview of the supply and demand of the financial asset in question.

Additionally, the limit order book also provides valuable information about market conditions and helps traders make informed decisions. By reviewing the supply and demand of an asset as represented in the limit order book, traders can gain insights into market trends and make decisions about their own trading strategies. This information can also be used by algorithmic traders to make high-frequency trading decisions based on real-time market data.

In summary, the limit order book is a crucial component of financial markets as it promotes fairness, transparency, and efficiency. It provides market participants with access to the best available prices and valuable market information, enabling them to make informed trading decisions.

## 2.2 Market making

Market making is a highly relevant financial trading strategy that is implemented to increase liquidity in a market by placing limit orders (LOs) at both sides of the limit order book (LOB). By doing so, market makers aim to profit from the spread of the transaction, which is the difference between the bid and ask price of a security. In other words, they buy a security at the bid price and sell it at the ask price, earning the spread as their profit.

Market making can be performed independently by a trader, with incentives provided by the exchange, or even by the exchange itself. The benefits of market making are mutual for both the market maker and the exchange. The increased liquidity provided by the market maker improves exchange execution speed and price, benefiting the exchange, while the market maker profits from the spread of their transactions.

It is common for exchanges to incentivize market making by offering a refund on a percentage of the market maker's transaction costs. This further encourages market makers to provide liquidity to the market, as they are able to offset some of their costs.

Market making requires a high level of market knowledge, risk management skills, and quick execution speed to be successful. The market maker must be able to determine the best prices for their LOs and react quickly to changes in market conditions. In addition, they must be able to balance their positions and manage their risk effectively, as they are exposed to market volatility while holding securities in both directions.

However, like all limit orders, the orders placed by market makers are not guaranteed to be fully filled at the specified price. As a result, market makers may accrue some non-zero inventory over the trading horizon. This inventory can generate significant losses due to price movements in the underlying asset, which is known as inventory risk. While inventory risk can result in profit, it is outside the core strategy of profiting from the spread and is therefore something that market makers aim to minimize.

For example, if a market maker places a buy order for 10 shares at a price of \$9.5 and a sell order for 10 shares at a price of \$10.5, and the buy order is fully filled for 10 shares but the sell order is only half-filled for 5 shares, the market maker has received a profit of \$5 from buying and selling 5 shares on each side with a spread of \$1 each. However, they now also hold a long position of 5 shares at \$10.5, with a portfolio value of \$52.5. If the price of the stock drops by \$2 during this period, the new value of the position is \$42.5, resulting in a loss of \$10 and a net loss of \$5.

Market making, while offering the potential for profit through the bid-ask spread, also entails various risks that must be acknowledged and carefully managed to minimize losses and optimize gains. Inventory risk is a prominent source of risk in market making and is a result of the buildup of holdings in the underlying asset due to unfulfilled or partially executed orders. If a market maker purchases more shares than they sell or vice versa, they will hold an inventory position that can result in monetary losses in the event of unfavorable price movements in the underlying asset. To reduce the impact of inventory risk, some market makers may pursue a net-zero inventory position at the conclusion of the trading day.

The second source of risk is execution risk, which occurs when orders do not execute immediately or at the desired rate. For instance, if a market maker places a limit order to buy shares at a certain price, but the price moves higher before the order is executed, the market maker may end up buying the shares at a higher price than intended. Execution risk can also occur if orders on one side are executed at different rates than orders on the other side, leading to imbalanced positions and potentially unfavorable market conditions.

The third source of risk is adverse selection risk, which arises from market conditions influencing order book behavior. For example, if there is an announcement of fraud at a company, this may lead to a decrease in buying activity, causing a market maker to accumulate an unfavorable inventory position. Adverse selection risk can also occur when market participants are aware that a market maker is looking to unwind a position and take advantage of this by lowering the prices they are willing to pay for the underlying asset.

## 2.3 Reinforcement learning

Reinforcement Learning (RL) is a subfield of Artificial Intelligence (AI) and machine learning (ML) that focuses on developing algorithms that enable an agent to learn how to make decisions based on trial and error in an environment. The learning process involves the agent receiving rewards or penalties based on its actions, with the goal of maximizing cumulative reward over time. RL algorithms have found applications in a wide range of areas, including gaming, robotics, autonomous systems, and finance.

In RL, the environment is modeled as a Markov Decision Process (MDP), where the state of the environment at each time step depends only on the previous state and the action taken by the agent. The agent interacts with the environment by selecting actions based on its current state, which then results in a new state and a reward. Over time, the agent learns to make optimal decisions based on its experience.

RL algorithms can be classified into two categories: model-based and model-free. Model-based RL algorithms rely on an explicit representation of the environment's dynamics, while model-free algorithms do not require a model and instead learn directly from experience. Model-free algorithms can be further divided into value-based methods, where the agent learns a value function that estimates the expected future reward for each state, and policy-based methods, where the agent learns a policy that maps states to actions directly.

RL has been applied to various financial applications, including algorithmic trading, portfolio management, and market making. By modeling financial markets as MDPs and training agents to optimize reward, RL has shown promising results in achieving improved trading performance compared to traditional methods.

A Markov Decision Process (MDP) is a mathematical framework used in reinforcement learning to model an agent's decision making process. It consists of the following components:

- A set of states $S$, where each state $s_i \in S$ represents the current environment or situation of the agent.
- A set of actions $A$, where each action $a_j \in A$ represents a choice that the agent can make in each state.
- A transition function $T(s, a, s')$, which describes the probability of transitioning from state $s$ to state $s'$ after taking action $a$. This function is defined as $T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$
- A reward function $R(s, a, s')$, which defines the immediate reward received after transitioning from state $s$ to state $s'$ by taking action $a$.

The goal of the agent in an MDP is to find an optimal policy $\pi^*$ that maps each state to an action, such that the expected cumulative reward over a horizon of time is maximized. This can be represented mathematically as:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$$

where $\gamma$ is the discount factor, which determines the weight given to future rewards.

## 2.4 Constrained reinforcement learning

Constrained Markov Decision Processes (CMDPs) are an extension of traditional Markov Decision Processes (MDPs) that introduce constraints on the actions that an agent can take in a given state. A CMDP can be mathematically represented as a tuple $(S, A, P, R, \gamma, C)$:

- $S$ is the set of states, representing the various situations in which the system can be.
- $A$ is the set of actions, representing the decisions that can be made in each state.
- $P_{s'|s,a}$ is the transition probability from state $s$ to state $s'$ given action $a$.
- $R_s^a$ is the reward obtained from taking action $a$ in state $s$.
- $\gamma \in [0, 1]$ is the discount factor, representing the importance of future rewards.
- $C$ is a set of constraints, where each constraint is a function $f_i(s, a) \leq 0$ specifying a limit on the sum of rewards that can be received.

$$\begin{aligned} \pi^*(s) &= \arg\max_{\pi} \sum_{t=0}^{\infty} \gamma^t r_{t+1} \\ \text{s.t.} \quad & s_{t+1} = T(s_t, a_t, w_t) \\ & a_t = \pi(s_t) \\ & b_t \leq B \end{aligned}$$

The goal of the CMDP is to find a policy $\pi(s)$ that maps each state $s$ to an action $a$, so as to maximize the discounted sum of rewards subject to the constraints:

Where $\gamma \in [0, 1)$ is the discount factor, $r_{t+1}$ is the reward at time $t + 1$, $T$ is the transition function that maps the current state and action to the next state, $w_t$ is a random disturbance, $a_t$ is the action taken at time $t$, $\pi(s)$ is the policy that maps states to actions, $b_t$ is the constraint function, and $B$ is the upper bound on the constraints.

The introduction of constraints allows for the representation of real-world situations where an agent's actions may be limited by external factors such as regulations or resource availability. In turn, this allows for more realistic and relevant modeling of decision-making scenarios, making CMDPs a useful tool for solving problems in fields such as finance, operations management, and control systems.

The key challenge in solving a CMDP is finding a policy that maximizes the expected cumulative reward while respecting the constraints. This can be approached through various methods, including linear programming, dynamic programming, and reinforcement learning.

## 2.5 Reinforcement learning algorithms

### 2.5.1 Q-learning

Q-learning is a reinforcement learning algorithm that is used to find the optimal action-value function for an agent in an environment. The Q-value for a state-action pair $(s, a)$ represents the expected cumulative reward that the agent would receive by taking action a in state s and following a fixed policy thereafter. The Q-learning algorithm updates the Q-values iteratively based on the observed rewards and the estimated Q-values of the subsequent states. The algorithm begins with an initial estimate for the Q-values, and then updates these estimates after each time step by using the following update rule:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max(Q(s',a')) - Q(s,a))$$

where $\alpha$ is the learning rate, r is the observed reward after taking action a in state s, $\gamma$ is the discount factor, $s'$ is the next state after taking action a in state $s$, and $a'$ is the action with the highest estimated Q-value in state $s'$. The algorithm continues updating the Q-values until they converge to the optimal action-value function.

The optimal action-value function represents the best possible expected cumulative reward that the agent can receive by taking actions optimally in each state. This function can be used to determine the optimal policy for the agent by selecting the action with the highest Q-value in each state. The Q-learning algorithm is a model-free algorithm, meaning that it does not require a model of the environment's transition dynamics. This makes the algorithm highly flexible and able to adapt to changes in the environment over time.

Q-learning has been used in a variety of applications, including robotics, game playing, and decision making. Despite its simplicity, Q-learning has been shown to be highly effective in solving complex problems in these domains.

### 2.5.2 Deep Q-learning

Deep Q-Learning is an advanced form of Q-Learning that leverages neural networks to approximate the Q-values for each action in a given state, instead of using a table as in regular Q-Learning. The neural network takes as input the state representation and outputs the Q-value for each action.

The key difference between Deep Q-Learning and regular Q-Learning is that Deep Q-Learning can handle high dimensional state spaces, whereas regular Q-Learning requires a discretization of the state space. This discretization results in a potentially large and sparse Q-table, which can be computationally expensive to update and may not accurately capture the complex relationships between the state and action values.

In Deep Q-Learning, the neural network acts as a function approximator, enabling the algorithm to learn and generalize the Q-values for a wide range of states, rather than being limited to the states it has seen in the past. The neural network can also capture non-linear relationships between the state and action values, which is difficult for traditional Q-Learning to do.

To update the parameters of the neural network in Deep Q-Learning, the algorithm uses the Bellman equation to calculate the target Q-value for each action, and uses this target to train the neural network to minimize the difference between the predicted and target Q-values. This process is repeated until convergence, at which point the neural network will have learned an accurate approximation of the optimal Q-values for each state-action pair.

### 2.5.3 PPO

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that is used to optimize the policy in a reinforcement learning problem. It combines elements from both value-based and policy-based methods and aims to strike a balance between exploration and exploitation in the optimization process.

The algorithm uses a value function to estimate the expected reward for each action and updates the policy accordingly, so as to maximize the expected reward. The objective of the algorithm is to find a policy that maximizes the following expression:

$$J(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where:

- $\theta$ represents the policy parameters
- $r_t(\theta)$ is the ratio of the new policy $\pi_\theta$ to the old policy $\pi_{\theta_{old}}$
- $\hat{A}_t$ is the advantage estimate for timestep $t$
- $\epsilon$ is a hyperparameter that defines the range for the clipping function clip

The algorithm updates the policy by taking gradient steps with respect to the objective function $J(\theta)$ using a variant of stochastic gradient ascent. The clipping function ensures that the updated policy does not deviate too far from the previous policy, and helps to stabilize the optimization process.

PPO is considered to be more sample-efficient and stable compared to other policy-based methods, and is widely used in deep reinforcement learning applications.

# 3 Methods

## 3.1 Data

The dataset used for this study comprised of order book L2 data and executed trades on the Binance platform for various cryptocurrencies from the period of 6 months from

March 2021 to September 2021 with sub-second granularity.

The following table provides the relevant columns and data types for the order book data:

| timestamp | ask[0].price | ask[0].amount | bid[0].price | bid[0].amount | ... |
|---|---|---|---|---|---|
| **Data Type** | float | float | float | float | ... |
| **Description** | Date & Time | Ask price at level 0 | Ask amount at level 0 | Bid price at level 0 | Bid amount at level 0 |

And here is the relevant columns and data types for the trades data:

| timestamp | kind | price | amount |
|---|---|---|---|
| **Data Type** | datetime | string | float |
| **Description** | Date & Time | Type of order (Ask or Bid) | Trade price |

## 3.2 Experimental design

### 3.2.1 Training environment

The training environment is a data driven simulator of a cryptocurrency exchange based on real market data provided by a third party vendor. Our approach is to train directly on historical tick level data from the order book. The various parts of our environment are as follows

- *State.* State of the environment will be the complete set orders (price, volume) on the exchange as well the statistics of the agent. This includes thing such as available cash and inventory amounts at different price levels.
- *Actions.* At every time step $t$ our agent with be able to submit $n$ tuples of both bid and ask limit orders $LO(P, N)$.

- *Rewards.* The agent will receive a reward based on the total number of trades it completes as well as the profit it generates. Overall exchange operators are encouraged to have tight spreads in order to prevent their customers from going elsewhere or losing money due to arbitrage.

State of the environment will be the complete LOB (price, volume) on the exchange as well other values about the underlying and the agent. This would include factors such as available cash and inventory amounts at different price levels. Exchanges operate on ticks for time $t$ meaning that we can formulate the objects as a discrete time MDP. As we know there will be some variable delay due to inference the time steps will be $t \in t_n 1, t_n 1, ..., t_n k$ with $n1, n2, ..nk$ being the number of ticks it requires for the agent to process and submit an order.

Along with discrete time increments every exchange has a minimum order price difference, causing the possible set of orders to be discrete as well. The action of placing a set of LO on either the buy or sell side can be represented as a sorted list similar to the definition of the LOB above.

### 3.2.2 Reward functions

A common approach in reinforcement learning for finance is to set rewards for the agent with the sole aim of maximizing profits. However, in our case, we need to directly incentivize the agent to minimize risky behavior. To do this, we add a penalty to the reward function that discourages large changes in the trading position.

At each time step $t_n$, the reward function can be formulated as follows:

$$R_n = \alpha \cdot PL_n + \beta \cdot V_n - \gamma \cdot I_n - \phi \cdot \sigma_{PL}$$

where:

- $PL_n$ is the one-step realized profit or loss
- $V_n$ is the current inventory value, represented by $Z_n(S_{n+1} - S_n)$ where $Z_n$ is the inventory level and $S_{n+1} - S_n$ is the change in price
- $I_n$ is the change in inventory, represented by $\Delta Z_n \Delta S_n$

- $\alpha, \beta, \gamma$ are constants that represent the weighting of each of these factors and can be adjusted to fine-tune the reward function

At the end of the trading day, an additional term is added to the reward function to account for the final time step. This is an inventory penalty term, represented by:

$$R_{EOD} = R_{EOD-1} - \rho \cdot Z_{EOD}$$

where $\rho$ is a parameter that determines the scale of the penalty based on the amount of inventory remaining. The value of $\rho$ can be adjusted to control the strength of the penalty for holding inventory at the end of the day.

–>

## References