Market Making via Reinforcement Learning

Alban Sarfati and Hooman Halabichian

CentraleSupélec, 91190 Gif-sur-Yvette, France
https://github.com/asarfa/MM_via_RL

Abstract. Market making is a fundamental trading problem in which an agent continually quotes two-sided markets for securities, providing bids and asks that he is willing to trade. Market makers provide liquidity and depth to markets and earn their quoted spread if both orders are executed. They are compensated for the risk of holding securities, which may decline in value between the purchase and the sale. The accumulation of these unfavourable positions is known as inventory risk. In this paper, we develop a high-fidelity limit order book simulator based on historical data with 5 levels on each side, and use it to design a market making agent using temporal-difference learning. We use a Deep Q-learning as a value function approximator and a custom reward function that controls inventory risk.

Keywords: Limit order book · Market Making · Inventory Risk · DQN.

1 Introduction

December 1998, in response to the technology boom sweeping the market, the US Securities and Exchange Commission (SEC) approved new rules that allowed electronic trading systems to register as full-fledged exchanges. Since then, algorithmic trading has become increasingly common as the need to process more data and act on shorter time scales makes the task almost impossible for humans.

This paper considers the problem of a market maker agent operating in an order-driven market. In such markets, matched orders result in trades and unmatched orders are stored in a limit order book, which is divided into two parts, a set of buy orders called bids, and a set of sell orders called asks.

In this context, the agent observes the current states of the environment (limit order book, inventory, etc) and uses this information to simultaneously and repeatedly posts limit orders on both sides at prices derived from an action-space table which affects the next states of the environment and in returns he receives a reward which is a risk-adjusted function based on its executed orders. His goal is to learn to act in a way that will maximize its risk-adjusted return measure over time.

The challenge for the market maker is to mitigate the inventory risk that comes from trading with more informed traders. This is because market makers are exposed to adverse selection, a phenomenon in which the market maker's counter parties exploit an advantage (technological or informational) when transacting with them. In particular, this causes the market maker to amass a (positive

or negative) inventory, before an adverse price move causes the market maker to incur a loss on this inventory.

1.1 Related Work

Market making has been a subject of investigation across several fields, including economics, finance, and artificial intelligence. In the mathematical finance literature, the conventional approach to market making has been to consider it as a stochastic optimal control problem. This involves developing models for order arrivals and executions, followed by designing and solving control problems for them [1,2,3,4]. The focus is often on analytically proving optimal or near-optimal control policies, which leads to restrictive action spaces for the market maker. Consequently, in many cases, the market maker only has control over a single order on each side of the market, and therefore, a single spread. In our paper we choose an action space, a trading strategy, a reward function and a state representation as in Spooner et al. [5]. They considered a realistic market simulator, using 5 levels of order book data, along with transactions. This was the first paper to train reinforcement learning agents on the vast quantities of so-called Level II data now available and to use a finite pre-specified selection of actions. The authors experimented with different state representations, reward function formulations and learning algorithms and coalesced the best performing components into a single agent exhibiting superior risk-adjusted performance. In their framework there was a slight partial observability problem: when the order book changes, and a transaction doesn't occur, it is not possible to know from where in the queue this cancellation/deletion came from. They had to assume a uniform distribution of such cancellations. The main novelty of this paper is to solve this observability problem, explore alternative learning algorithms and looked at basic form of inventory control.

1.2 Our contribution

The purpose of this paper is to use reinforcement learning in order to design competitive market making agents for financial markets using high-frequency historical equities data. The steps taken to develop our agents are outlined below
(1) We developed a very realistic order book simulator, using historical data from LOBSTER that is combined with our agents' orders.
(2) We used neural networks to represent the policy and value functions allowing to handle non-linear patterns, tackle low signal-to-noise ratios and manage large state, which are associated with financial data.
(3) We looked at inventory management. Using market orders to control inventory results in a cost, therefore we developed agents skewing dynamically their limit orders to favour driving the absolute value of inventory to zero.

2 Preliminaries

2.1 Limit order books

A limit order book (LOB) is a record of outstanding limit orders maintained by the security specialist who works at the exchange. A limit order is a type of order to buy or sell a given amount of a security at a specific price or better. A market order is a request to buy or sell now at the current market price. The market spread, s, is the difference between the best bid and ask quoted in the LOB. The mid-price, m, is the mid-point of the spread. When the best bid or ask changes, the mid-price move, denoted by Δm, which is the change in the mid-price since the last time period.

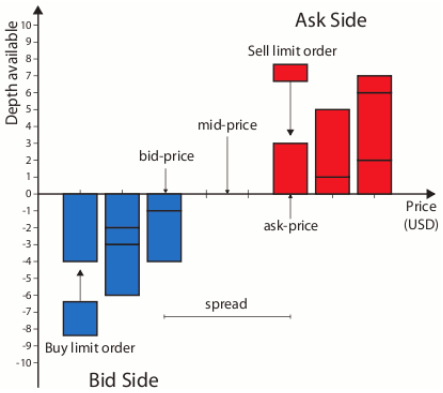

Fig. 1. Graphical representation of the Limit Order Book.

2.2 Data

We developed a high-fidelity reconstruction of the limit order book using high-frequency (up to nanoseconds) historical data. These historical data are provided by LOBSTER which is an online tool furnishing easy-to-use, high-quality limit order book data derived from NASDAQ's Historical TotalView-ITCH. LOBSTER gives access to sample files for multiple tickers (MSFT, AAPL, AMZON, etc) on 21 June 2021 evolution between 09:30:00 and 15:59:59. These files are composed of a 'messages' and an 'orderbook' file. The 'orderbook' file contains the evolution of the LOB up to the requested number of level (5 in this work). The 'message' file contains indicators for the type of event (order) causing an update of the LOB in the requested price range. The type of event can be: 1) Submission of a new limit order, 2) Cancellation (partial deletion of a limit order), 3) Deletion (total deletion of a limit order). Each event is characterized by an id, a size (volume), the price and the direction (buy or sell). Table 1 and 2 give a snapshot of these processed files.

	ask_price_1	ask_size_1	bid_price_1	buy_size_1	ask_price_2	ask_size_2
2012-06-21 09:30:00.013994120	30.99000	3788	30.95000	300	31.05000	200
2012-06-21 09:30:00.015247805	30.99000	3788	30.95000	300	31.04000	100
2012-06-21 09:30:00.015442111	30.99000	3788	30.95000	300	31.04000	100
2012-06-21 09:30:00.015789147	30.99000	3788	30.95000	300	31.04000	100
2012-06-21 09:30:00.016299574	30.99000	3788	30.95000	300	31.04000	100
2012-06-21 09:30:00.020733480	30.99000	3788	30.95000	300	31.03000	100
2012-06-21 09:30:00.022011407	30.99000	3788	30.95000	300	31.02000	100
2012-06-21 09:30:00.022343661	30.99000	3788	30.95000	300	31.01000	100

Table 1. Orderbook file processed.

	event	id	size	price	direction	ticker
2012-06-21 09:30:00.013994120	submission	16116348	100	31.05000	sell	MSFT
2012-06-21 09:30:00.015247805	submission	16116658	100	31.04000	sell	MSFT
2012-06-21 09:30:00.015442111	submission	16116704	100	31.05000	sell	MSFT
2012-06-21 09:30:00.015789147	submission	16116752	100	31.06000	sell	MSFT
2012-06-21 09:30:00.016299574	submission	16116815	100	31.07000	sell	MSFT
2012-06-21 09:30:00.020733480	submission	16118193	100	31.03000	sell	MSFT
2012-06-21 09:30:00.022011407	submission	16118930	100	31.02000	sell	MSFT
2012-06-21 09:30:00.022343661	submission	16119035	100	31.01000	sell	MSFT

Table 2. Message file processed.

2.3 Simulator

In order to create a simulator which tracks the top 5 price levels in the LOB and allows the agent to place its own orders within these price levels, we developed programming-oriented objects modelling the different backbones. In doing so, the simulated exchange is highly realistic and makes limited assumptions about the market, allowing the agent to interact with the order book directly: certain properties of real order books naturally arise.

Database The processed data from the 'orderbook' and 'message' files are stored in a 'HistoricalDatabase' object, which allows to retrieve the information at specific time.

Orders The orders figuring in the processed 'orderbook' and 'message' of the database are transformed to an object depending of their nature, resulting in market, limit, cancellation or deletion order. As an example, the first row of Table 2 becomes: LimitOrder(timestamp=Timestamp('2012-06-21 09:30:00.01399412'), direction='sell', ticker='MSFT', externalid=16116657, isexternal=True, price=31.05, volume=100).

LOB The limit order book is as well modelled as an object, by aggregating the limit orders into buy-sell SortedDict allowing to compute properties such as best buy price, spread, midprice etc.

Fig. 2 represents the first LOB object i.e. the one characterized by the information in the first row of Table 1, while Fig. 3 allows to visualise it.

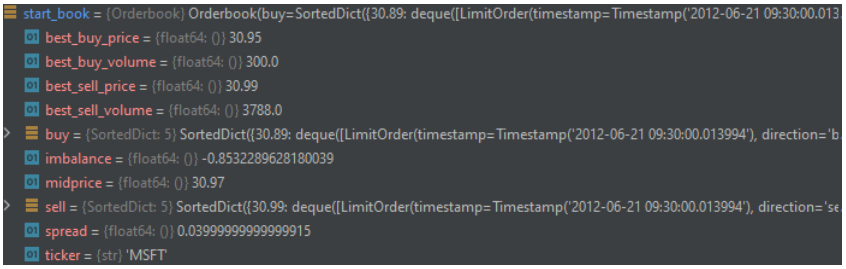

Fig. 2. First LOB object.

This starting LOB is modified over time from the historical orders in the 'messages' file and the agent's orders, characterized by isexternal=False, that are generated.

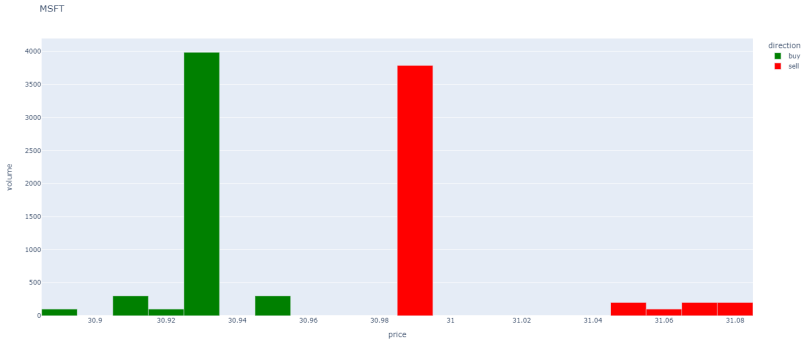

Fig. 3. First LOB plot.

OrderGenerator A class 'HistoricalOrderGenerator' is developed to generate a list of historical orders objects from a starting date to an ending date. Note that these historical orders are derived from the original 'messages' file.

Exchange A class 'Exchange' allows to create the initial LOB object, based on the first snapshot of the orderbook in the database, and from there to process each new order, thanks to bid and ask data from the messages in the database. - Each limit order is submitted to the LOB if it doesn't cross the spread, otherwise it is executed. A buy (sell) limit order cross the spread when its price is superior (inferior) to the best sell (buy) price. If the order price is equal to an initial level, the order is added to the queue of that level otherwise the order constitutes a new level. If the number of levels is higher than 5, we can either

leave it as it or delete the worse orders to return to 5. However, some order's volume can be very large which results in taking all the liquidity from the LOB. Hence, we didn't delete the worse orders but our agent can only place orders in the first 5 levels (as the action-space will show).
- Each market order is executed at the best limit order. A buy (sell) market order is executed at the best sell (buy) price given that this specific limit order has enough volume otherwise it will execute against the remaining best limit orders. Therefore, for each order executed one or more limit order will be filled (reduced or cleared) from the LOB given queue positioning.
- Each cancellation and deletion order removes the respective limit order from the LOB given queue positioning.

Fig. 4, allows to visualize the Exchange representation at time step 1.

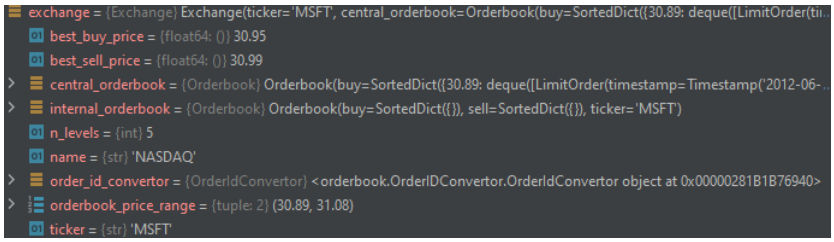

Fig. 4. Exchange object.

Simulator Given the previous classes, the main one, 'OrderbookSimulator' allows to reset the Exchange arguments for each episode according to a start date, and then do a forward step until an end date: if the agent acts every one second, then the forward step lasts one second. This forward step method takes into account a list of orders from the agent and from the OrderGenerator until the end date, and call the Exchange to process all of these orders one by one. This results in two lists of FilledOrders if some orders have been executed against historical limit order or/and agent's limit order. One issue arise at the end of the forward step when the modified LOB price range has exited the initial LOB price range i.e some limit buy (sell) orders prices are lower (higher) than the initial minimum buy (maximum sell) price. The new levels have to be updated directly from the orderbook in the database corresponding to the same timestamp, in order to reflect the true volumes of these deepest levels. Fig. 5 is a view of our Simulator object.

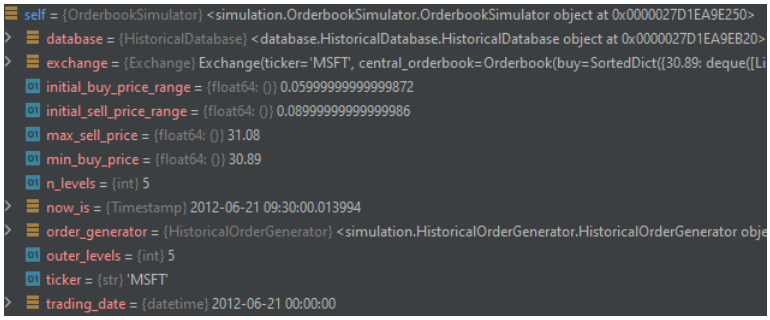

Fig. 5. Simulator object.

3 Environment

3.1 Trading strategy

The market making agent acts every second on events as they occur on the LOB. An event is anything that constitutes an observable change in the states of the environment (change in mid price, spread, best buy price, etc). The agent is restricted to a single buy and sell order, per second, with an order size of 100, and cannot exit the market. Given that the market is built based on past data, any orders simulated by the agent would not have any impact on the market. Hence, we chose a small order size of 100, which in comparison to the overall trading volume of the market has a negligible impact. The available limit orders actions are represented in Table 3.

$ActionID$	0	1	2	3	4	5	6	7	8
$Ask\,(\theta_{\mathrm{a}})$	1	2	3	4	5	1	3	2	5
$Bid\,(\theta_{\mathrm{b}})$	1	2	3	4	5	3	1	5	2

Table 3. Action-space for market making.

These actions are limit orders to be placed at fixed distances relative to a reference price, $Ref\,(t_i)$, chosen here to be a measure of the true market price: the mid-price m. At each time step, t_i, the agent revises two control parameters, $\theta_{\mathrm{a}}\,(t_i)$ and $\theta_{\mathrm{b}}\,(t_i)$, from which the quoting distances, $Dist_{\mathrm{a,b}}\,(t_i)$, are derived

$$Dist_{\mathrm{a}}\,(t_i) = \theta_{\mathrm{a}}\,(t_i) \cdot Spread\,(t_i)$$

$$Dist_{\mathrm{b}}\,(t_i) = \theta_{\mathrm{b}}\,(t_i) \cdot Spread\,(t_i)$$

Where $Spread\left(t_i\right)$ is calculated by taking the market half-spread: $s\left(t_i\right)/2$. The agent's pricing strategy, is given by these two equations

$$p_{\mathrm{a}}\left(t_i\right) = Ref\left(t_i\right) + Dist_{\mathrm{a}}\left(t_i\right)$$

$$p_{\mathrm{b}}\left(t_i\right) = Ref\left(t_i\right) - Dist_{\mathrm{b}}\left(t_i\right)$$

The smaller the values of $(\theta_{\mathrm{a}}, \theta_{\mathrm{b}})$ leads to buy-sell quotes that are closer to the top of the book, which increases their likelihood of being executed. Actions with ID $\in (5, 8)$ allow to skew the limit orders in favour of a side.

As an example, let's suppose that the current mid-price is USD 100 and the spread is USD 2, if the agent choose the action 0 then it will post a buy limit order at the best buy price which is USD 99 and post as well a sell limit order at the best sell price: USD 101. If both orders are executed at a given time, the agent will capture the spread i.e will earn USD 2, while providing liquidity to the market. However, if only one of the limit orders is executed the agent not only fail to capture the quoted spread but obtains a non-zero inventory and take on the inventory risk that arises from fluctuations in the market value of the asset held.
Hence, the main source of risk for a market maker comes from holding important inventory, the amount of stocks currently owned or owed, during adverse price swings. Therefore, a market maker must make sure their inventory remains within some reasonable bounds, to do so, one of the options is to place a market order to liquidate some of its position in the asset. Hence, an action is added to Table 3, represented by ID = 9, which allows to create a market order that brings the agent closer to a neutral position if the inventory at time step t_i denoted $Inv(t_i)$, constraints are no longer satisfied. These constraints are defined as the maximum absolute inventory that the agent is willing to hold. The market order is sized proportionately to the agent's current inventory, $Size_{\mathrm{m}}\left(t_i\right) = -\alpha \cdot Inv\left(t_i\right)$, where $\alpha \in (0, 1)$ is a scaling factor. However, the agent pays a cost for this risk management in the form of crossing the spread.

To model this action-space, a class 'OrderDistributor' is developed allowing to convert an action (characterized by its ID $\in (0, 8)$) to the sell and buy prices according to the control parameters of the ID. Action 9 is considered independently.

Inventory-driven policy An optimal market-maker agent should skew its bid and ask quotes in the opposite direction to its asset holdings. As an example, if it holds a negative inventory, it should skew its limit orders in favor of buying, which makes it more likely that it will get filled on the bid-side that brings its inventory closer to zero, helping him to complete round-trip trades and encouraging mean-reversion of its inventory. Actions with ID $\in (5, 8)$ allow to execute this strategy. Unlike the market order placing strategies of action 9, the inventory-driven agent manages to do this without crossing the spread and incurring a cost.

3.2 Reward function

An idealised market maker seeks to make money by repeatedly making its quoted spread, while keeping its inventory low to minimise market exposure. The "natural" reward function for trading agents is the PnL which represents how much money is made or lost through exchanges with the market.

At each time step, t_i, $Matched_{\mathrm{a}}(t_i)$ and $Matched_{\mathrm{b}}(t_i)$ are the amount of volume matched (executed) against the agent's orders since the last time step t_{i-1} in the ask and bid books, and $m(t_i)$ denotes the mid-price at time step t_i.

$$\psi_{\mathrm{a}}(t_i) = Matched_{\mathrm{a}}(t_i) \cdot [p_{\mathrm{a}}(t_i) - m(t_i)]$$

$$\psi_{\mathrm{b}}(t_i) = Matched_{\mathrm{b}}(t_i) \cdot [m(t_i) - p_{\mathrm{b}}(t_i)]$$

These functions compute the money lost or gained through executions of the agent's orders relative to the mid-price. The PnL function $\Psi(t_i)$ is given by

$$\Psi(t_0) = 0$$

$$\Psi(t_i) = \psi_{\mathrm{a}}(t_i) + \psi_{\mathrm{b}}(t_i) + Inv(t_i)\,\Delta m(t_i)$$

PnL:

$$r_i = \Psi(t_i).$$

This basic formulation of PnL, captures both the gain from speculation and the gain from spread, which can lead to instability during the learning, due to inventory risk. The inventory term is the leading driver of agent behaviour and should be manipulated to match one's risk profile. To this end, another PnL function have been engineered to disincentive trend-following, enhance spread capture and model risk aversion.

Asymmetrically dampened PnL:

$$r_i = \Psi(t_i) - \max[0, \eta \cdot Inv(t_i)\,\Delta m(t_i)].$$

This reward function, is encouraging market making behaviour by reducing the reward that the agent can gain from speculation, thank to the dampening factor η which reduces only the profits from speculation.

To model these rewards, we created a class 'Portfolio' having 3 attributes: 1) inventory, 2) cash, 3) gain. The attributes of this class are updated each time agent's orders have been filed. For each limit buy (sell) order filled the portfolio inventory is updated by adding (subtracting) the order's volume, the portfolio cash is updated by subtracting (adding) the order's volume times the order's price. The portfolio gain at each time, t_i:

$$\psi_{\mathrm{a}}(t_i) + \psi_{\mathrm{b}}(t_i)$$

$$= buyvolume * (buyprice - midprice) + sellvolume * (midprice - sellprice)$$

Based on these elements the chosen reward and the PnL is computed at each step. The comparison of the cumulative reward and the cumulative PnL makes it possible to distinguish the agent's current financial situation, based on current market conditions, from the chosen dampened PnL function, which models its risk aversion.

3.3 Sate representation

The environment is composed of two states, the agent-state and the market-state. These states are constructed as set of attributes that describe the agent and the market. At each time step, t_i:
- The agent state is composed of $Inv(t_i)$ and $Dist_{\mathrm{a,b}}(t_i)$.
- The market state is composed of $s(t_i)$, $\Delta m(t_i)$.
Moreover, technical indicators are derived from the orderbook attributes and added to the market state, they represent statistical tools and are extensively use to make investment decisions by generating signals. They help to identify trends, regime switches, momentum, and potential reversal points. These technical indicators are Book/queue imbalance, Volatility, Relative strength index, Trade direction imbalance, Trade volume imbalance.

Three different representations of state are considered, the agent-state, the market state and full-state.

To model these features, we created a class 'State' having 5 attributes: 1) filled orders, 2) orderbook, 3) midprice 4) portfolio 5) control parameters, which are updated at each step. Furthermore, we developed a class for each one of the features allowing to reset the values, update them and normalise (MinMaxScaler([-1, 1])) them based on the attributes of class 'State'.

3.4 Environment

Given the previous elements, a class 'OrderbookEnvironment' is developed in order to represent the environment. This class has two main methods:
- Reset(): Constructs the initial OrderDistributor, Simulator, Portfolio, State and Features.
- Step(action): Converts the action took by the agent into orders by using OrderDistributor. Uses the method forward step of Simulator and update the Portfolio, State and Features according to the evolution of data. Calculates the reward accordingly to the action took. Checks if the episode is done and computes information such as rewards statistics, actions took, performance criteria, etc.

4 Agent

4.1 Policy

Market prices are Markovian in nature, i.e. the probability distribution of the price of an asset depends only on the current price and not on the prior history. Hence, it makes sense to formulate this problem as a Markov Decision Process (MDP) problem with the goal being to solve this MDP by finding an optimal policy. For the agent, the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$ describes the problem:
- $\mathcal{S}$ is the set of states.
- $\mathcal{A}$ is the set of actions.
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability distribution which determines how the environment changes based on the state and actions taken by the agent.

- $r : \mathcal{S} \times \mathcal{A} \to R$ is the reward function.
- $\gamma \in (0, 1)$ is the discount factor which determines the importance of future rewards.
Let R_t denote the sum of future rewards, i.e.

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T$$

Then, at time-step t, the future discounted reward is then given by

$$R'_t = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \ldots + \gamma^T \cdot R_T = \sum_{i=t}^{T} \gamma^i \cdot R_{t+i+1}$$

However, states in the context of market making are a great source of complexity, the market state is in constant evolution, a multi-agent system and data can be unrepresentative (black swan). Moreover, the states encountered are most often not the complete states since there exist many other traders in the environment. Thus, the market making problem is a Partially Observable Markov Decision Process (POMDP). The state the agent observes, $\mathcal{S}'$, is some derivation of the true state, $\mathcal{S}$, i.e. $\mathcal{S}' \sim \mathcal{O}(\mathcal{S})$. As there is a potentially infinite number of states that our agent could be in, we need a function approximation method to help the agent deal with this.

The algorithm used is deep Q-learning. The goal of reinforcement learning algorithms is to find an optimal policy $\pi : \mathcal{S} \to \mathcal{A}$, i.e. a function mapping between states and actions such that it maximizes the expected long-term reward, known as the Q-value. Q-learning is a model-free value-based approach, which uses the action-value function $Q : \mathcal{S} \times \mathcal{A} \to R$, where $s \in \mathcal{S}$ and $a \in \mathcal{A}$, to estimate the expected reward given some state-action pair. The policy that maximizes the rewards is given by

$$\pi(s) = argmax Q(s, a)$$

Under this policy π, the true value of being in state s and performing action a is given by

$$Q^\pi(s, a) = E_\pi \left[R_t \mid \mathcal{S}_t = s, \mathcal{A}_t = a, \pi \right]$$

Neural networks with their universal approximation capabilities are a natural choice to find a function $Q_\theta(s, a)$ that approximates $Q^\pi(s, a)$, the input are the states of the environment and the output are the Q-values for each of the actions.

The target Q-Value should be close to the reward the agent gets after playing action a in state s plus the future discounted value of playing optimally from then on.

$$Q_{target}\left(s_t, a_t\right) = r_{t+1} + \gamma \max_a Q_\theta\left(s_{t+1}, a\right)$$

s_t is the state at time-step t, a_t is the action taken at state s_t, r_{r+1} is the direct reward of action a_t, γ is the discount factor, and $\max_a Q_\theta\left(s_{t+1}, a\right)$ is the maximum delayed reward given the optimal action from the current policy Q_θ.

The difference between the two sides of the last equality is known as the temporal difference error, δ :

$$\delta = Q_{target}\left(s_t, a_t\right) - \left(r_{t+1} + \gamma \max_a Q_\theta\left(s_{t+1}, a\right)\right)$$

4.2 Target network

It's important to note that training a Deep Q-Network (DQN) for market making is quite challenging due to the complexity, dynamic and volatility of the market, thus it is difficult to accurately estimate the value of an action in the long term. A target network is a separate copy of the Q-network that is used to generate target Q-values during training. The weights of the target network are frozen and updated periodically with the weights of the Q-network. This creates a more stable target for the Q-learning algorithm to learn from, as the target Q-values are not constantly changing during training. Additionally, using a target network helps to prevent the agent from overfitting to the current state of the environment and chasing short-term trends, which can lead to poor performance in the long term.

4.3 Exploration-Exploitation Trade-off

There is a trade-off between exploration (trying new actions) and exploitation (using the best known action) for an agent. To balance this trade-off, a decaying ϵ-greedy approach is commonly used, where the agent takes random actions with a probability of ϵ and takes the best known action with a probability of 1-ϵ, while ϵ decreases over time. This approach encourages the agent to explore more at the beginning and exploit more at the end, meaning it will stick to the best-known action in the later stages.

4.4 Experience Replay

Experience replay involves storing an agent's experiences in a memory buffer and randomly sampling batches of experiences to reduce temporal correlation in the data. This allows the agent to relive past experiences and learn from them, even as the market shifts. Training on these randomly sampled batches helps the agent to converge towards the optimal policy and update the parameters of its function approximator using stochastic gradient descent. By using experience replay, the agent can learn more efficiently and adapt to changes in the environment.

4.5 Performance criteria

The PnL which measures the total wealth of the agent at the end of the episode is computed. To assess if an agent has taken a speculative approach in trading (large inventories), the mean absolute position (MAP) held by the agent is quoted as well. Moreover, a custom rolling PnL-to-MAP Ratio is designed,

which simultaneously considers both the profitability and the incurred inventory risk of the agent strategy up to the time t. The uncertainties of the PnL and the absolute position (AP) per time-step are quoted using the mean absolute deviation.

4.6 Agent

The DQN agent is based on two shallow neural networks (policy and target). The networks take in as inputs the state as seen in 3.3. and outputs 9 Q-values which are then used to determine the optimal action. In Fig. 6 a depiction of the training framework is provided.

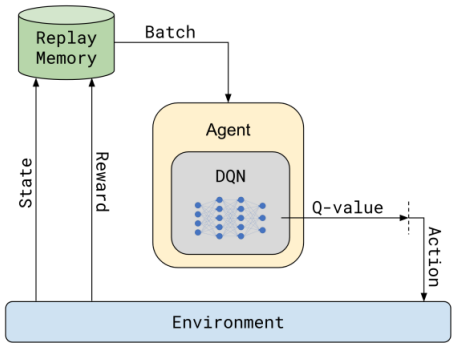

Fig. 6. DQN-agent Training Framework.

The details of how training happens within this framework are outlined in Fig. 7.

Note Once the agent places an order, the match engine attempts to match the agent's order to orders on the opposing order book side. Only when there is a matching order does a trade happen. Therefore, the goal of the agent is to select the best actions according to its state (current inventory level, market signals, etc). This is a challenging optimization problem since trades happen at random time and there are many unknown variables:
- Other Market Participants: This agent is not the only one trading on the market. There are many other market participants that the agent interacts with.
- Adversarialness: Among these market participants, there are also algorithmic traders, some of which might be playing adversarially.
Moreover, our agent is designed to leave its orders open even if they haven't been executed after a certain time. However, if at any time certain orders are no longer competitive, they are cancelled.

Initialize replay memory M and batch size
Initialize $\epsilon, \epsilon_{decay}, \epsilon_{min}$ and discount factor γ
Initialize Q-network with random weights
Initialize target network as a copy of Q-network and τ
Initialize n=0, the number of total steps
for episode = 1 to 1000 **do**
    $s_t \leftarrow$ Reset environment (environment outputs initial state)
    while not done **do**
        $n \leftarrow n + 1$
        $a_t \leftarrow$ Select random action with probability ϵ or choose action $\max_a Q_\theta(s_t, a)$
        $s_{t+1}, r_t, done, info \leftarrow$ Act based on a_t
        if replay memory M is full **then**
            Remove the first element from M
        end if
        Append $(s_t, a_t, r_t, s_{t+1}, done)$ to replay memory M
        if $n\%100 == 0$ **then**
            $b \leftarrow$ Sample mini-batch from replay memory M at random
            $q \leftarrow$ Initialize empty array of size $|b|$
            for each $(s_i, a_i, r_i, s_{i+1}) \in b$ **do**
                $q_i \leftarrow r_i + \gamma \cdot \max_a Q_\theta\left(s_{i+1}, a_i\right)$ if not done else r_i
            end for
            Apply gradient descent: Loss$(q_i, Q_\theta(s_i, a_i))$
        end if
    end while
    Evaluate agent on testing environment
    Decay ϵ
end for

Fig. 7. Deep Q-Learning Training.

5 Experimental setup

For the purposes of our research we trained different version of our agent on the Microsoft sample and compare it to fixed and random "spread-based" strategies. Each learning agent is trained and evaluated in the respective Microsoft training and testing environment for 1000 episodes and we report the results in 6. for the best episode on the testing environment. Hence, for this best episode the agent is saved allowing us to evaluate it on other stocks, like Google, on the same testing window.

Trading Strategy We consider either using the market order with the order clearing factor $\alpha = 0.99$ with a maximum absolute inventory of 10 000 or letting the agent skew its limit orders in favor of one side as seen in the inventory-driven policy.

Reward function The asymmetric dampening PnL function is found to produce superior risk-adjusted performance in most cases as it lead to improved stability during learning and better asymptotic performance thanks to the dampening factor η. We chose $\eta = 0.1$ and $\eta = 0.7$, representing a low risk-averse versus a high risk-averse agent.

State representation The full-state is rich enough to contain all the relevant information and sparse enough to enable quick learning, which is essential to facilitate learning and improve the efficiency of the algorithms. Hence, we consider only the full-state representation.

Environment The time-series dataset is split into a training and test set with a proportion of 0.8, where all of the testing data occurs chronologically later than the training data. Hence, the training environment starts randomly between 10h and ≈14h and lasts 30 minutes. During the evaluation, the environment starts at ≈14h30 and lasts until 15h30. This single interaction is defined as one episode. We ignore the first half and last half hour of trading because they are not suited for our task as it's when the market opens and closes, respectively, and tends to be more volatile and less informative for our task.

Policy We set γ to 0.97, as it allows to give more weight to future rewards and the goal is to maximize the long-term performance. To minimise δ, we use the Huber loss. The Huber loss acts like the mean squared error when the error is small, but like the mean absolute error when the error is large - this makes it more robust to outliers when the estimates of Q are very noisy.

Target Network The Q-network and target network are updated every 100 steps, the soft update for the target network is controlled by $tau = 0.01$, which helps to improve the stability and performance of the learning process.

ϵ-greedy policy As the environment is large and the task is complex, we used $\epsilon = 0.99$, $\epsilon_{decay} = 0.99$ and $\epsilon_{min} = 0.01$. These values encourage the agent to explore during 400 episodes, allowing to improve its policy and adapt to changes in the environment by discovering new, potentially better actions that it may not have encountered before. Hence, the last 600 episodes are based only on exploitation.

Experience replay The size of the memory buffer is set to 1e6, as the size of the dataset is large and the problem is complex. This large memory buffer can help the agent to learn more efficiently, effectively explore the state-action space, and improve its performance in the long run. The batch size is set to 512, as a high size should provide sufficient information for updating the model parameters from the memory buffer.

DQN The networks are composed of Linear() or LSTM() layers. Linear() layers are followed by ReLU() activation. In the case of LSTM() layers, the activation function used is Tanh() and the state covers the last 60 time-steps (one time-step = one second) to represent the sequential nature of the problem. For both types, a dropout() layer with probability = 0.1 is added before the hidden layer. The

numbers of neurons per layer is 256 in order to capture the complexity of the problem. Furthermore, the Adam optimizer, with its default parameters, is used for training with a learning rate of 0.001.

Note All these parameters have not been tuned on a validation environment due to the computational cost it would have involved, but were set according to our point-of-view and the literature.

5.1 Strategies

A fixed strategy is represented by taking only one kind of action all the time while a random strategy takes action randomly according to randint() and seed 42, these benchmark strategies are set without market clearing order.
Different learning agents have been trained with variations on the usage of market clearing order or inventory-driven policy, the factor η of the reward (asymmetric dampened PnL function) and the DQN type of layers:
-HDLinear: inventory-driven, η=0.7, layers=Linear()
-SDLinear: inventory-driven, η=0.1, layers=Linear()
-HDLstm: inventory-driven, η=0.7, layers=LSTM()
-SDLstm: inventory-driven, η=0.1, layers=LSTM()
-MCLstm: market clearing, η=0, layers=LSTM()

6 Results

6.1 Data analysis

On the 2012-06-21 trading day, one can observe from Fig. 8 that there were 592 184 events causing an update of the LOB. The volume order had a mean of 546, a min of 1 and a max of 200 000. The majority of order types were submission and deletion. We can observe that there was more selling limit order (26%) than buying limit order (23%). This is typical of a bear market with more asks than bids. Only 5% of events represent market orders.

	size
count	592184.00000
mean	546.43136
std	960.57514
min	1.00000
25%	100.00000
50%	300.00000
75%	700.00000
max	200000.00000

	Frequency event per direction
cancellation/buy	0.00469
cancellation/sell	0.00406
deletion/buy	0.21233
deletion/sell	0.23336
execution/buy	0.02536
execution/sell	0.02496
submission/buy	0.23208
submission/sell	0.26316

Fig. 8. Descriptive statistics Messages.

The downward trend, represented in Fig. 9, in the mid price confirms the bear market of MSFT on 2012-06-21. Moreover, Fig. 10 shows that the imbalance

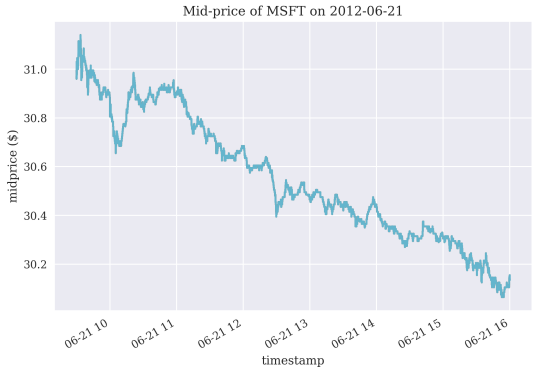

Fig. 9. Mid price.

volume histogram is skewed towards the sell side.

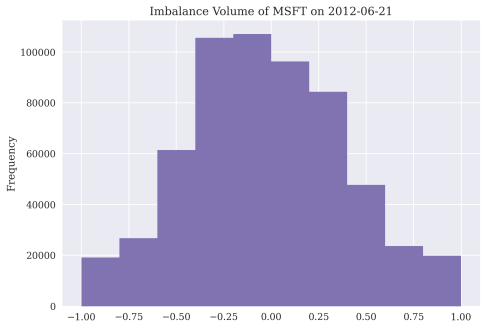

Fig. 10. Imbalance volume.

6.2 MSFT

From Table 4, one can see that all agents have a negative PnL on Microsoft testing environment (2012-06-21, ≈14:30 - 15:30). The agent reaching the maximum PnL is our SDLstm with a final PnL of -1134$ and a MAP of 50567. Fixed strategies with $\theta_{a,b} \in [1,5]$ have an increasing MAP as $\theta_{a,b}$ increases, while Fixed 7, which skew it's limit orders in favor of the sell side, has better out-of-sample PnL than our learning agents composed of Linear() layers. Indeed, as seen in Fig. 9, the mid price has a downward trend, hence, a speculative agent can chose

to have a negative inventory, creating profit by short selling. Action 7 allows to so, with limit sell order at the top of the book that are executed with more likelihood than limit buy order deep in the book. Moreover, the extra action of placing a market order with the clearing factor $\alpha = 0.99$ and the maximum absolute inventory of 10 000, allows to achieve substantial lower MAP as we can see with our MCLstm agent but unfortunately doesn't lead to the best PnL, one cause being that the agent pays a cost for this risk management.

Strategy	PnL (dollars)	MAP (units)
Fixed 0($\theta_{\mathrm{a,b}} = 1$)	-7370	37136
Fixed 1($\theta_{\mathrm{a,b}} = 2$)	-7346	45168
Fixed 2($\theta_{\mathrm{a,b}} = 3$)	-9643	58791
Fixed 3($\theta_{\mathrm{a,b}} = 4$)	-11732	75588
Fixed 4($\theta_{\mathrm{a,b}} = 5$)	-9683	77275
Fixed 5($\theta_{\mathrm{a,b}} = 1, 3$)	-3755	38389
Fixed 6($\theta_{\mathrm{a,b}} = 3, 1$)	-13264	66789
Fixed 7($\theta_{\mathrm{a,b}} = 2, 5$)	-1564	55909
Fixed 8($\theta_{\mathrm{a,b}} = 5, 2$)	-15583	83075
Random	-8908	54641
HDLinear	-3607	45452
SDLinear	-3271	46199
HDLstm	-1554	48347
SDLstm	$-\mathbf{1134}$	50567
MCLstm	-1446	3906

Table 4. Out-of-sample performance criteria on MSFT.

Fig. 11 shows the evolution of the PnL and inventory of our different agents on Microsoft testing environment. One can see, that the performance criteria of our agents with the inventory-driven policy have the same trend. Their PnL have an upward trend until 15:10, at which time their inventory starts to be highly positive, resulting in losses. The inventory of our agent with the market clearing order fluctuates around ± 10000, which makes its PnL curve much more stable, but with a downward trend throughout the test.

Although our SDLstm agent has the highest average PnL per-step (-0.3), its mean absolute deviation (26.2) is higher than that of our HDLstm agent (25.6), resulting in lower stability, as depicted in Table 5. Our MCLstm have the lowest mean absolute deviation for PnL (2.5), which shows that holding small inventories is a contributing factor towards the reduced uncertainty on PnL.

As our agents have been trained and evaluated during 1000 episodes, we can observe the evolution of the testing performances (cumulative risk-adjusted reward and mean absolute position) of our SDLstm agent in Fig. 12, which relies on the exploitation of the optimal policy. The cumulative reward has an upward trend while the mean absolute position has a downward trend and for both criteria their variance is lower as the number of episode increases, hence catastrophic

Fig. 11. Evolution of performance criteria for different strategies on MSFT.

Strategy	PnL (dollars)	AP (units)
Fixed 0($\theta_{a,b} = 1$)	-1.9 ± 23.3	172 ± 323
Fixed 1($\theta_{a,b} = 2$)	-1.9 ± 28.1	167 ± 320
Fixed 2($\theta_{a,b} = 3$)	-2.4 ± 38.5	149 ± 288
Fixed 3($\theta_{a,b} = 4$)	-3.0 ± 47.5	136 ± 266
Fixed 4($\theta_{a,b} = 5$)	-2.4 ± 47.6	134 ± 260
Fixed 5($\theta_{a,b} = 1, 3$)	-0.9 ± 22.0	168 ± 320
Fixed 6($\theta_{a,b} = 3, 1$)	-3.4 ± 41.5	153 ± 291
Fixed 7($\theta_{a,b} = 2, 5$)	-0.4 ± 28.3	163 ± 314
Fixed 8($\theta_{a,b} = 5, 2$)	-3.9 ± 52.0	137 ± 265
Random	-2.3 ± 35.1	153 ± 288
HDLinear	-0.9 ± 26.7	162 ± 308
SDLinear	-0.8 ± 26.9	162 ± 308
HDLstm	-0.4 ± 25.6	165 ± 315
SDLstm	$\mathbf{-0.3 \pm 26.2}$	165 ± 315
MCLstm	-0.4 ± 2.5	16 ± 308

Table 5. Mean and MAD of out-of-sample per-step criteria on MSFT.

forgetting happens less often as the number of episodes increases. These graphics indicate that our algorithms worked well as a good market making strategy seeks a high PnL and a low inventory.

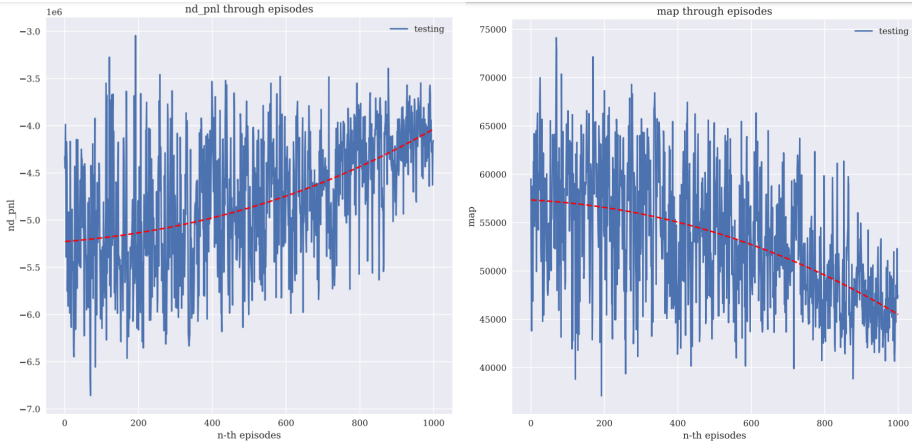

Fig. 12. SDLstm cumulative reward and MAP per episode.

Fig 14. in the Appendix represents the multiple metrics and behavior of our SDLstm agent at the best episode on Microsoft testing environment. One can note that by acting every second, the agent took 3958 actions, most of were $\theta_{\text{a,b}} = 2, 5$ and then by ascending order: $\theta_{\text{a,b}} = 1, 3$, $\theta_{\text{a,b}} = 3, 1$, $\theta_{\text{a,b}} = 2$, $\theta_{\text{a,b}} = 4$ and $\theta_{\text{ab}} = 1$. In doing so, our agent chose to skew its limit orders in favour of the sell side as he may have inferred that the mid-price had a downward trend. Moreover, only 5.2% of its orders were filled, and just 3 actions filled both side (bid, ask) during the same step size of 1 second. Hence, most of the time the inventory of the agent is neither small nor close to zero and the capture of the spread is difficult. Something intriguing, is that the buy limit orders at the level 5 and 3 of the book (deep) have been filled more often that the sell limit orders at the level 2 and 1 (top). This is why the agent's inventory went from negative to positive when it was not intended to do so, resulting in losses. The per-step risk-adjusted reward (dampened PnL with $\eta = 0.1$) has a mean of -2 with a standard deviation of 136, a min of -1946 and a max of 958, they do not represent a bell curve and are negatively skewed. Last but not least, the PnL-to-MAP ratio of our agent is non-linear with a high variance.

6.3 GOOG

To assess whether our agents generalise well to the test environment (2012-06-21, 14:30 - 15:30) of other stocks, we chose Google, whose type of activity is similar. One can see, from Table 6, that most of the agents have this time a positive

PnL, and our HDLstm is the one with the highest final PnL (55283$), followed by Fixed 7 whose PnL is equal to 32408$.

Strategy	PnL (dollars)	MAP (units)
Fixed 0($\theta_{a,b} = 1$)	1812	83137
Fixed 1($\theta_{a,b} = 2$)	6257	89396
Fixed 2($\theta_{a,b} = 3$)	10706	85304
Fixed 3($\theta_{a,b} = 4$)	32861	83250
Fixed 4($\theta_{a,b} = 5$)	7895	79941
Fixed 5($\theta_{a,b} = 1, 3$)	5672	99451
Fixed 6($\theta_{a,b} = 3, 1$)	-157	78517
Fixed 7($\theta_{a,b} = 2, 5$)	32408	102862
Fixed 8($\theta_{a,b} = 5, 2$)	-8372	68601
Random	8899	83013
HDLinear	3916	85705
SDLinear	-34528	81293
HDLstm	**55283**	102392
SDLstm	17588	90103

Table 6. Out-of-sample performance criteria on GOOG.

The PnL evolution of our HDLstm agent has an upward trend and outperforms that of other agents most of the time, as Fig. 13 shows. One factor could be that it's inventory is always more negative than that of other agents, which exposes him more to changes in the value of the stock, resulting in speculative trading. One should find this quit intriguing as the agent used a high dampening factor ($\eta = 0.7$) which should lead to smaller positions. However, as described above the agent has been trained only on Microsoft, therefore the risk-adjusted reward has only an effect on the MSFT policy and doesn't affect evaluation.

By looking at the metrics of our HDLstm agent for the best episode presented in Fig. 15, most of its actions were $\theta_{a,b} = 2, 5$ and then by ascending order: $\theta_{a,b} = 1, 3$, $\theta_{a,b} = 4$, $\theta_{a,b} = 3$ and $\theta_{ab} = 3, 1$. The agent's inventory is always negative, as the sell limit orders at level 2 and 1 are the most executed. 36.5% of its limit orders were filled, and more than 150 actions filled both side (bid, ask) during the same step size, leading to a better capture of the spread than with Microsoft. The per-step risk-adjusted reward (dampened PnL with $\eta = 0.7$) has a mean of 3 with a standard deviation of 458, a min of -27657 i.e a large penalty due to the high η and a maximum reward of 2233. Last but not least, the PnL-to-MAP ratio of the agent has a positive trend which is encouraging.

6.4 Overall

The results indicate that our DRL approach outperforms the benchmarks and achieves the most favorable PnL and MAP. Indeed, for the evaluation of Microsoft and Google stock our agents trained with LSTM() layers achieves respectively more than 27% and 70% higher terminal wealth (PnL) than the Fixed

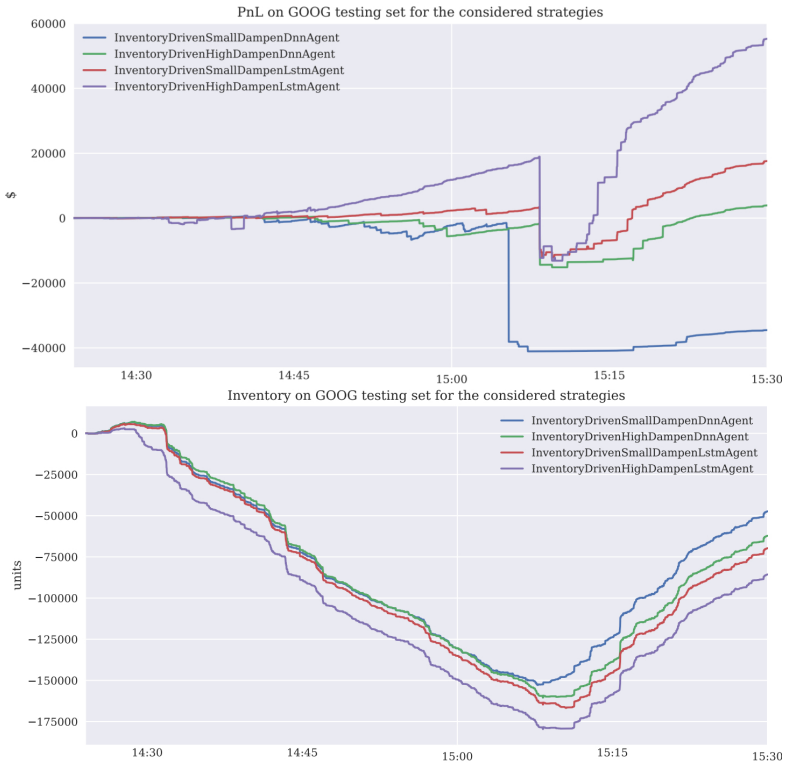

Fig. 13. Evolution of performance criteria for different strategies on GOOG.

$7(\theta_{\mathrm{a,b}} = 2, 5)$ while being exposed to less than 10% and 0.5% of its inventory risk (MAP). However, in these test environments, our agents hold large inventories at all times, exposing themselves to changes in the value of the stock over long periods of time, which leads to the noise (volatility) observed in the PnL curve. These consistently high mean absolute positions can be explained by the fact that our agents have chosen to skew their orders towards the sell side: $\theta_{\mathrm{a,b}} = 2, 5$ and $\theta_{\mathrm{a,b}} = 1, 3$ constitute more than 90% of the actions took, to exploit clear downward trends, whilst simultaneously capturing the spread. Note that an optimal market making policy trained on more data, not just that representing this bear market, would target a near-zero inventory, which would lead to a more stable PnL and would use actions with ID $\in (5, 8)$ to encourage mean-reversion of inventory and not speculative trading. We can also observe the small performances of our agents trained with Linear() layers, which are less profitable and more volatile, as they are not suited to the sequential nature and high complexity of the problem. As seen with the Microsoft results, increasing the dampening factor η, seems to reduce the MAP, this reduction in exposure, comes along with a change in PnL, which also tends towards lower variance as a result. Hence, increasing values of η should yield better and more consistent performance.

7 Conclusions

Several agents based on a deep reinforcement learning framework were provided to solve the problem of optimizing market making. The results showed that the policy these agents were able to learn led to competitive out-of-sample performance and demonstrates superior performance over several standard Market Making benchmark strategies. The application of reinforcement learning to such a problem also shows that it is capable of working well in complex environments and that it is a viable tool that can be used for market making. We believe that our framework can be improved as our agents lack of proper inventory management in the MSFT, GOOG test environments and could use less speculative trading to achieve the consistency that is expected from an optimal market making strategy. The following directions for future research seem promising:

• Training the agents over several stocks, several trading days and a larger number of episodes to ensure better generalisation properties.

• Expand to a larger number of LOB levels with consequent actions, as it would allow an agent to skew its limit orders deeper in the book, increasing the likelihood to execute only one side, which would encourage mean-reversion of its inventory.

• Bayesian optimization would be a natural approach to tune the parameters of the framework.

• Extend to multi-order and variable order size action spaces.

• Investigate the impact of market frictions such as fees and latency on the agent's strategy.

• More emphasis should be placed on explaining the resulting RL policies, an area that has been little explored so far, despite financial regulators' requirements for explanation.

• All current reinforcement learning approaches assume stationarity. Therefore, the issue of non-stationarity of financial markets is still not addressed. The development of adaptive RL agents that can adapt to dynamic and noisy environments by continuously learning new dynamics and gradually forgetting the old ones could be a way to solve this problem.

References

1. Frédéric Abergel, Marouane Anane, Anirban Chakraborti, Aymen Jedidi, and Ioane Muni Toke. 2016. Limit Order Books.
2. Marco Avellaneda and Sasha Stoikov. 2008. High-frequency trading in a limit order book. Quantitative Finance 8, 3 (2008), 217–224
3. Álvaro Cartea and Sebastian Jaimungal. 2015. Risk Metrics and Fine Tuning of High-Frequency Trading Strategies. Mathematical Finance 25, 3 (2015), 576–611
4. Joseph Jerome, Leandro Sánchez-Betancourt, Rahul Savani, and Martin Herdegen. 2022. Model-based gym environments for limit order book trading. arXiv preprint arXiv:2209.07823 (2022).
5. Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. 2018. Market Making via Reinforcement Learning. In Proc. of AAMAS. 434–442

8 Appendix

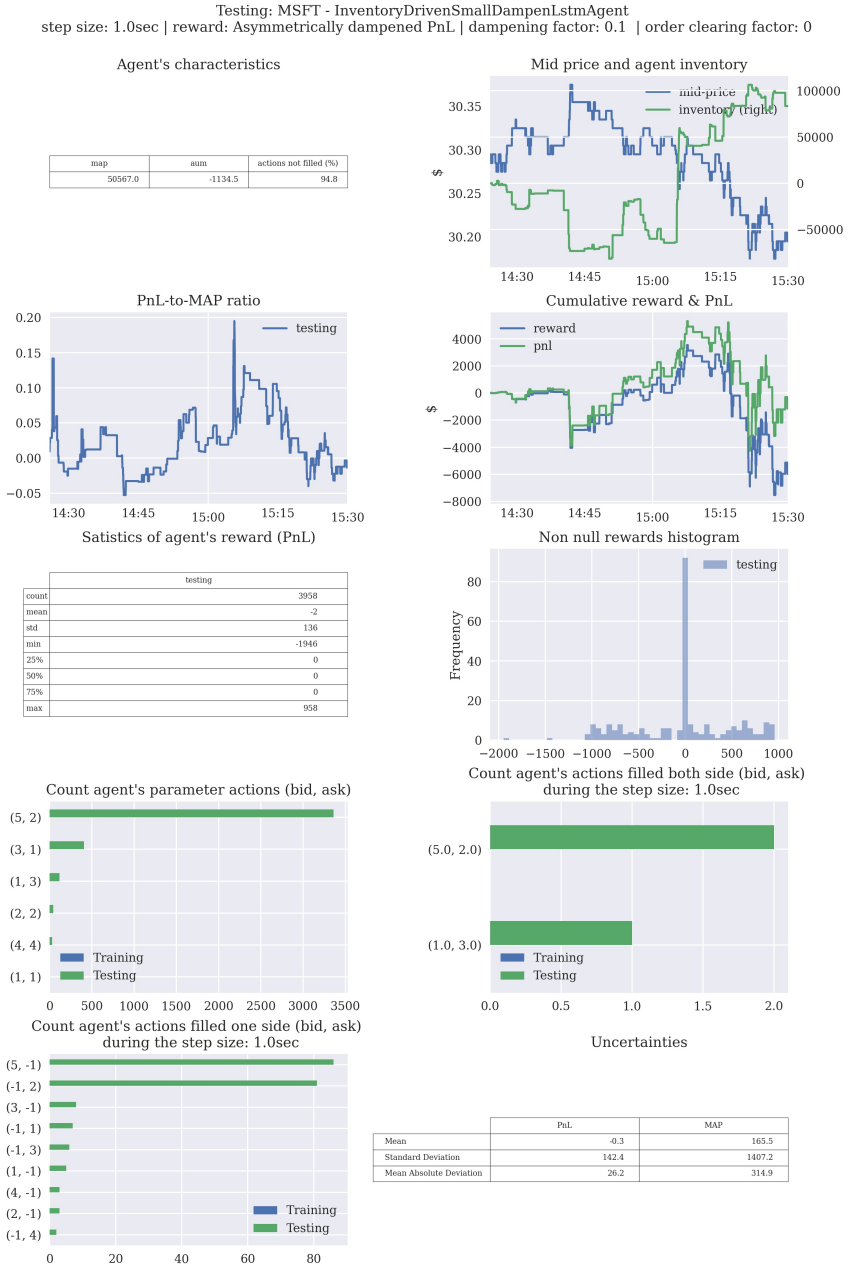

Fig. 14. SDLstm testing metrics for best episode on MSFT.

Fig. 15. HDLstm testing metrics for best episode on GOOG.