Deep Reinforcement Learning for High-Frequency Market Making

Pankaj Kumar PK.MPP@CBS.DK
Copenhagen Business School, Denmark

Editors: Emtiyaz Khan and Mehmet Gönen

Abstract

High-frequency market making is a algorithmic trading strategy in which an agent provides liquidity at the same time as quoting a bid price and an ask price on a security. The strategy reap profits in the form of the spread between the quoted price placed on the buy and sell prices. Due to complexity in inventory risk, counterparties to trades and information asymmetry, the understanding of high-frequency market making algorithms is relatively unexplored by academics across disciplines. In this paper, we develop realistic simulations of limit order markets and use them to design a high-frequency market making agent using Deep Recurrent Q-Networks. Our approach outperforms a prominent benchmark strategy from literature, which uses temporal-difference reinforcement learning to design market making agents. Using the simulation framework, we analyse how the maker-take fee, a feature of market design, affects market quality and the agent's profitability. The agents successfully reproduce stylised facts in historical trade data from each simulation.

Keywords: High-Frequency Trading; Deep Reinforcement Learning; Agent-Based Models.

1. Introduction

The electronification of securities trading has transformed traditional human-driven markets into predominantly automated ones, in which high-frequency trading (HFT, or high-frequency traders) typically exceeds 80% of total volume traded in US listed equities (McGroarty et al., 2019; Menkveld, 2013). HFT is a form of automated trading in which security positions are turned over very quickly by leveraging advanced technology and associated extremely low latency rates Menkveld (2016). High-frequency market making is an HFT-based strategy that contributes to market liquidity by matching buyer and seller orders. The profit is earned from the spread between the quoted price placed on the buy and sell prices. Given the ever-growing minuscule limit order book (LOB) data, complexity in inventory risk, counterparties to trades and information asymmetry, the current understanding of high-frequency market making algorithms is relatively shallow Avellaneda and Stoikov (2008); Abernethy and Kale (2013); Spooner et al. (2018). This paper uses a variant of Deep Recurrent Q-Networks (DRQN) to design high-frequency market making agents that interact with a realistic limit order book simulation framework.

1.1. Related Literature

A number of high-frequency or classical market making strategies have been proposed across disciplines, including finance Avellaneda and Stoikov (2008); Chakraborty and Kearns

(2011), econophyics McGroarty et al. (2019) and machine learning Brahma et al. (2012); Abernethy and Kale (2013); Spooner et al. (2018). Earlier work in the field of finance considers market making as a problem of stochastic optimal control, in which order book dynamics are designed using control algorithms after developing the arrival and execution model (Avellaneda and Stoikov, 2008; Cartea et al., 2014) to understand the price impact, adverse selection, risk measures and inventory constraints.

Another prominent approach consists of agent based models (ABM, or agent-based modelling), ranging from zero intelligence to intelligent variants. These are used to study market making, but are typically evaluated in simulated markets without using real market data. This gives the modeller the flexibility to churn out potentially emergent phenomena as a result of interaction between agents. The evolution of technology- based disruptions in HFT mean that the existing learning models and empirical models are deficient and may no longer be appropriate. Reinforcement learning has been applied for market making (Spooner et al., 2018), algorithmic trading (Wei et al., 2019), optimal execution (Nevmyvaka et al., 2006) and foreign exchange trading (Dempster and Leemans, 2006). However, defining hand-crafting features in reinforcement learning for agents to learn while interacting within a dynamic environment is a major stumbling block. Also, reinforcement learning could be slow to learn in large state spaces, and the methods are not generalisable (e.g. across the state space).

Deep learning eliminates the need for manual feature design, and therefore the need to find compact representations in high-dimensional data. It also helps to generalise across states, thereby improving sample efficiency for large state-space reinforcement learning problems. Augmenting deep learning with reinforcement learning, i.e. deep reinforcement learning (DRL), enables reinforcement learning to scale up to problems with high-dimensional state and action spaces. The outstanding success stories of DeepMind, with its ability to achieve superhuman-level performance in Atari 2600 video games proves the effectiveness of DRL. However, only a few examples have addressed optimal execution (Ning et al., 2021), market making (Elwin, 2019) and high-frequency trading (Wei et al., 2019), rather than games.

The success of these single DRLs can be attributed to the use of experience replay memories, which enable Deep Q-Networks (DQNs) to be trained efficiently through sampling stored state transitions. However, despite the ever-increasing performance on popular benchmarks such as Atari 2600 games, DQNs struggle to generalise when evaluated in different environments. They do not perform well in partially observable domains (Hausknecht and Stone, 2015), they overestimate action values under certain conditions (Hasselt et al., 2016) and are not efficient when it comes to prioritising experience replay (Schaul et al., 2016). The proposed Deep Recurrent Q-Networks (DRQN) (Hausknecht and Stone, 2015) using recurrent neural networks – in particular, LSTM (Long Short-Term Memory) – solve the above problem by replacing the first post-convolutional fully connected layer with an LSTM layer in a DQN setting. The incorporation of this layer means that DRQN has sufficient memory capacity to work with only one input, rather than a stacked input of consecutive frames. Double DQN (Hasselt et al., 2016) obliterates the overestimation problem in DQN, resulting in more stable and reliable learning. By prioritizing experience, Schaul et al. (2016) achieved a new state-of-the-art in human-level performance across benchmark Atari games. We incorporate this innovation in DRL to design a high-frequency market

making agent, investigate the interaction of the agents with others agents in the market, and study how this interaction affects market quality under different market designs.

1.2. Contributions

This paper's main contribution is the development of a realistic simulation of limit order markets, which is then used to design a high-frequency market making agent that employs DRQN. We modify the classical DQRN architecture and incorporate double Q-learning and prioritised experience to take account of volatile, illiquid and stagnant markets. Our approach outperforms a prominent benchmark strategy from literature, which uses temporal-difference reinforcement learning to design market maker agents. We enumerate our contributions as follows. We designed realistic simulations framework for limit order markets using a matching engine, communication interface and the Financial Information exchange (FIX) protocol. The simulation framework takes account of the agent's latency and the build-up of maker-taker fees , as defined in NYSE. We extended the agent strategy to take account of order size, adverse selection and cancellations. We proposed a reward function that takes account of transaction costs and maker-taker fees . We modified DRQN architecture with double Q-learning and prioritised experience to design high-frequency market making agents. We investigated the performance of DQN, DRQN and RL market making agents on our simulation framework. We analysed the effect of maker-taker fees on market quality using our simulation framework. We validated the simulation framework by reproducing stylised facts in historical trade data from each simulation.

2. DRL Simulation Framework

In this section, we describe the environment (simulation framework), agents (trading strategies), actions (buy, sell or cancels orders), states (market features), and rewards (profit and loss) to set the stage for reinforcement learning formulation for high-frequency market making. The agents interact with the equity market on limit order book events from 09:00 to 16:00, subject to inventory constraints. The occurrence of an event is due to a perceptible reconstruction in the state of the order book. As such, the agent's actions are inhomogeneous in time. Trading outside of regular hours was not allowed in our framework, but is performed through electronic communication networks known as dark pools, which match potential buyers and sellers without using a traditional stock exchange (Forde and Putniņš, 2015).

2.1. Environment: Simulation Framework

We have designed a simulation framework over realistic market design, market engine, communication interface, and the FIX protocol. This framework is unconstrained by historical data, represents realistic exchange and makes no assumptions about the market. From a high-level perspective, the simulation framework comprises three entities, agents a market and interaction mechanisms, as shown in Figure 1. Markets act as communication nodes that listen for agents to make connections and process incoming orders, which are aggregated in order books, and create trades according to a matching engine designed for each instrument, etc. Matching engines consists of high-capacity, low- latency order-matching

servers that provide the transactional probity for an electronic trading venue using various algorithms to facilitate the matching of buyers and sellers. The most common of these is price/time priority or First In First Out (FIFO). FIFO ensures that all orders at the same price level are filled according to time priority.

Figure 1: High-level simulation framework.

2.2. Agents: Trading Strategies

In our simulation framework, we populate the market with two types of agents: *high-frequency market makers* and *high-frequency market takers*. The high-frequency market makers provide liquidity by submitting multiple limit orders on both sides of the order book in order to capture profits from the bid-ask spread. In return for subsidising the provision of the liquidity, the high-frequency market makers typically receive a small rebate from the exchange upon execution of their orders. Conversely, the high-frequency market takers remove liquidity by placing market orders in the market, which is reinforced by the continuous availability of a tight bid-ask spread created by the high-frequency market makers. The exchanges charge the high-frequency market takers a nominal fee for their executed orders. The relationship between high-frequency market makers and high-frequency market takers is considered symbiotic, as neither can thrive without the other. The agents interact with the market via order type, price and quantity according to their internal logic. The submitted orders in the limit order book are matched using price-time priority algorithms from the matching engine. The latency manager in the simulation framework manages the whole history with suitable timestamps for orders to help the agents maintain a tight inventory. The two agents' trading strategies are discussed below.

2.2.1. High-Frequency Maker's Strategy

In this paper, we implement a realistic high-frequency market making strategy that takes account of the order size, which was absent from previous literature (Spooner et al., 2018). It is roughly based on the liquidity providing strategy described in a prominent research study by Karvik et al. (2018).

At each event time t, the total quantity of liquidity Q_t high-frequency market maker willing to provide a fixed proportion of their available capital C_t is defined as $Q_t = \omega C_t$. The high-frequency market maker's available capital, C_t, comprises of starting capital plus the profits accumulated from buy and sell trades up to time t, and the profit or loss from the remaining inventory holdings:

$$
\begin{aligned}
C_t = C_0 \\
&+ \min\left(\sum_{m,t-1} d_i^b - \sum_{m,t-1} d_i^a\right)\left(\frac{\sum_{m,t-1} d_i^a p_i^a}{\sum_{m,t-1} d_i^a} - \frac{\sum_{m,t-1} d_i^b p_i^b}{\sum_{m,t-1} d_i^b}\right) \\
&+ \left[\max\left(\sum_{m,t-1} d_i^b - \sum_{m,t-1} d_i^a\right) - \min\left(\sum_{m,t-1} d_i^b - \sum_{m,t-1} d_i^a\right)\right]\bar{p}_{t-1}
\end{aligned} \tag{1}
$$

where, m_t is the history of all trades in time t, d_t is liquidity demand in time t, $p_t^{a,b}$ is bid/ask price of a asset at time t and $\bar{p}_t$ is observed market mid-price at time t.

The limit order size that high-frequency market maker is willing to buy or sell is defined as:

$$
q_t^{a,b} = \left(\frac{1}{2}Q_t\right) \cdot \left(\frac{1}{N}\right) \cdot \left(\frac{I \pm \bar{I}}{\bar{I}}\right) \cdot \Upsilon\left(1 - \left|\frac{\bar{p}_{t-1}}{\bar{p}_{t-t_c}}\right|\right) \tag{2}
$$

where, I_t, $\bar{I}$ are inventory at time t and maximum inventory respectively; Υ for accounts for adverse selection and t_c is price change period. The total quantity of liquidity Q_t high-frequency market maker wishes to supply is equitably divided into buy and sell orders. These are then further subdivided into N distinct limit orders on each side of the order book. The splitting is done to adjust inventory at an optimum level. The risk of adverse selection is accounted for using the last term with Υ as a parameter. For detail discussion on the variables defined above is carried on in the research article (Karvik et al., 2018).

At each event time, t, high-frequency market maker update parameters, Θ_t^a and Θ_t^b, which is required for deriving relative prices, $\mathbb{D}_t^{a,b}$. The high-frequency market makers encounters adverse selection risk when posting limit orders at fixed distance from mid price $\bar{p}_t$. The volatility of mid prices in the previous five periods is chosen as proxy for the risk, with Λ governing the sensitivity of the bid-ask spread to volatility. The bid-ask spread is set as linear function of above volatility, subject to minimum constant, ξ. After setting bid/ask, further orders are placed either side of book above/below former price, using parameter τ. The equations below define the pricing strategy for the high-frequency market maker's limit order:

$$
p_t^{a,b,i} = \bar{p}_t + \mathbb{D}_t^{a,b} \pm \min\left(\Lambda\sigma_{(t-1:t-5)}, \xi\right) \pm i \cdot (10^{-\tau}), \quad \mathbb{D}_t^{a,b} = \Theta_t^{a,b} \cdot \mathbb{S}_t \tag{3}
$$

where spread, $\mathbb{S}_t$, is a moving average of the market half-spread (Spooner et al., 2018).

At each event time t, the high-frequency market making agents can clear their outstanding orders in the limit order book using two criteria. The first corresponds to clearing the inventory using a market order if it has not been executed at the end of trading. The second criterion takes account of the current market condition to remove the order from the limit order book. In particular, we define the probability of cancellation as $\Psi_t = 1 - \exp^{-\psi \cdot vol_p}$, where ψ is sensitivity parameter and vol_p is the perceived volatility (Bartolozzi, 2010). The high-frequency market maker follows simple heuristics to cancel orders in the limit order book. First, the range of ∓ 20 from the most recent transaction price is identified. Then,

the existing order is investigated for being outside of the price range. If there are orders outside the range, then the Ψ_t percent of the order will be cancelled, while the rest remains in the order book.

2.2.2. High-Frequency Taker's Strategy

As discussed above, the high-frequency market takers wishes to fill their trade immediately by agreeing with the currently listed prices on the order book. When trading large assets over the course of the day, the high-frequency market taker tends to minimize price impact and trading cost. In our simulation framework, high-frequency market taker follow a momentum strategy. This is the digital equivalent of classical day traders, who earn profits from market movements by aggressively taking liquidity. In this simple momentum trading strategy, the trend is captured using a price change rate, defined as:

$$\Delta p_t = \frac{p_{(t)} - p_{(t-t_c)}}{p_{(t-t_c)}} \tag{4}$$

where p_t is the price of the asset at time t and t_c is the price change period.

The size of the market order is proportional to the strength of the price rate change, subject to inventory constraints. In other words, the size of the market order will be:

$$d_{MK,t} = (\delta) \cdot (\Delta p_t) \cdot \left(1 - \left(\frac{I_{MT,t-1}}{\bar{I}_{MT}}\right)^h\right) \tag{5}$$

where δ is sensitivity of order size to price movement parameter, $I_{MT,t}$ is high-frequency market taker's inventory at time t, $\bar{I}$ is maximum inventory and h controls the order size as as $I_{MT,t}$ approaches to $\bar{I}$.

2.3. State Representation

The state representation comprises the *agent-state* and *market-state*, which contain information about agent's position as well as market features. The agent-state is defined by the following variables. Inventory at time t, I_t, which takes account of the amount of assets bought or sold by the high-frequency market making agent. The active quoting distances, normalized by spread, $\mathbb{S}_t$. The high-frequency market maker's update parameters, Θ_t^a and Θ_t^b, which is required for deriving relative prices, $\mathbb{D}_t^{a,b}$. The past price history, n_h, which is used to recognise the market trend or risk. The cancellations probability, Ψ_t, which is used to clear outstanding orders in the LOB.

The complexity of the market is represented by the market-state, which contains the partial observable state of the limit order book during each event period, as well as any prior information from previous periods. In this paper, we include the following market features, as described in benchmark paper (Spooner et al., 2018) and other. Bid-Ask spread, Mid-price move, Queue imbalance (Cartea et al., 2014), Volume imbalance (Weber and Rosenow, 2005), Orderbook depth (Weber and Rosenow, 2005), Signed volume, Perceived volatility (Bartolozzi, 2010) and Relative strength index.

2.4. Action Space

In our high-frequency market making setting, there are four possible actions between which the agent must decide: "buy", "hold", "sell" and "cancel". The agent can buy/sell fixed multiples of integer values at particular price $p_t^{a,b}$ at time t. The cancellation also can be done in only integer values. At the end of trading, the high-frequency market maker uses a market order to clear their inventory of anything that has not been executed or cancelled.

2.5. Reward Functions

In this paper, the reward function is traditional profit and loss (PnL), which keeps track of money gained or lost. The agents try to maximise the profits accumulated during a trading day, subject to inventory. To incorporate realism, as per the existing market design, the maker-taker fee model is also included. The maker-taker fee model is a pricing structure in which an exchange customarily pays its members a per- share rebate to supply (i.e. "make") liquidity, and charges them a fee to remove (i.e. "take") liquidity. For example, agents may be charged 0.0030 per share for taking liquidity from the market (i.e. 3 dollars per 1,000 shares) and receive a rebate of 0.0020 per share for posting liquidity (i.e. 2 dollars per 1,000 shares).

At a given event time t, lets us assume that high-frequency market making agents post buy/sell limit order of size $q_t^{a,b}$, he/she receives execution confirmation at time $t + \Delta t$. The Δt is vaguely referred as latency provided by the simulation framework, as there is always time a lag between a request made and actual transaction done. The much ignored transaction costs in academic literature is incorporated using an exponential penalty on the number of shares executed and maker-taker fee as defined in NYSE. Notably, we define the PnL function as:

$$\mathbb{R}_{PnL}(t) = \sum_{m_t} \left(q_t^a p_t^a - q_t^b p_t^b \right) - \alpha \left(\exp^{\frac{q_t^{a,b}}{\Delta t}} \right) \pm \beta \left(F_{(maker/takerfee)} \right) \tag{6}$$

3. Architecture and Algorithms

In this section, we introduce the modified DRQN architecture and algorithms to train the same, and briefly specify the agent's exploration/exploitation policy, which is aimed at maximising rewards.

3.1. DRQN Architecture

We minimally modify the existing DRQN architecture (Hausknecht and Stone, 2015) by replacing the fully connected network layers with LSTM layers, as shown in Figure 2. In this way, the architecture is adapted to handle partial observability and long-term order book data dependency, which is a common problem in market making. Next, we discuss an important module of the DRQN architecture to provide an overview of the process.

3.1.1. Basic Network

The states as input is fed to a convolutional neural network with three convolutional layers. The first layer contained 32 filter with a size of 8×8 and 4 stride. The second layers convolves the first layer output with 64 filter of a size of 4×4 and 2 stride and the last layers convolves the second layer output with 64 filter of size 3×3 and 1 stride, respectively. To prevent vanishing gradients, we use a rectifier nonlinearity activation function (ReLU) activation function at each convolutional layers. The output of the convolutional neural network is then passed into fully connected LSTM layers, followed by a nonlinear rectifier too. The LSTM layers by combining current observation o_t and history information h_{t-1} gives output $O(o_t, h_{t-1})$. This is then used to approximate the Q-value $Q(o_t, h_{t-1}, a_t)$. In the next time step, the history information was updated using $h_t = LSTM(o_t, h_{t-1})$ and passed through the hidden state to the network. After that, the output of LSTM layers is transformed to Q-value $(Q(s,a)_t)$ tensor for each possible action at next position, using fully-connected linear layer.

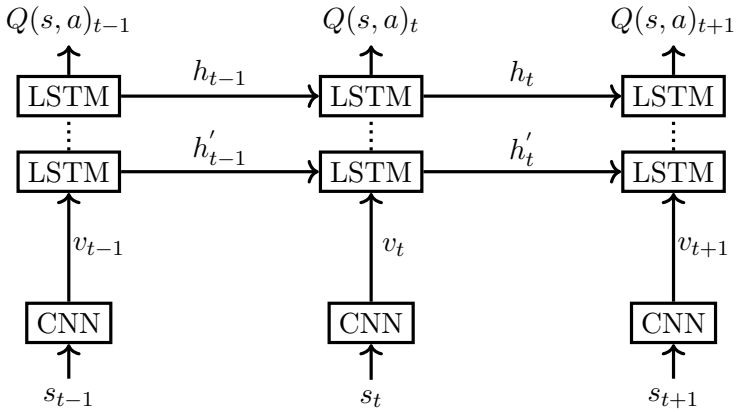

Figure 2: The Deep Recurrent Q-Network.

3.1.2. Target Network

At every step of training, the same network $Q(s, a|\theta)$ is used to calculate next state's target Q-values and updates the current state's Q-values. Consequently, the value of $Q(s^{'}, a^{'}|\theta)$ changes while changing θ. Thus, the network might suffer from oscillations in the gradient updates, which could even obstruct learning entirely. To prevent this instability, the target network ($Q(s, a|\theta_T)$) is used together with main network $Q(s, a|\theta_M)$. In this case the target network is fixed while updating main network. The parameters used to calculate the target value are frozen for a set number of iterations and then replaced by θ_M only after several periods of trading (Dijk, 2017; Ning et al., 2021).

3.1.3. Prioritised Experience Replay

The experience replay proposed by Dijk (2017) straightforwardly replays the originally experienced transitions without weighting them in terms of significance, which might compromise the efficiency of the algorithms.In the paper (Schaul et al., 2016), the authors discuss two

different ways of sampling from the replay buffer: rank-based prioritisation and proportional prioritisation. We use the former, which we consider more robust in terms of outliers. In addition, the heavy-tail property of rank-based prioritisation guarantees sample diversity, which is important for agents learning efficiently while trading in volatile markets. For a detailed discussion of the resulting algorithm design, performance, comparison, robustness and scalability, please refer to the original paper by Schaul et al. (2016).

3.2. Algorithm

The detailed procedure to train DRQN agents is summarized in Algorithm 1. We start by initializing the experience reply $\mathcal{D}$ of size $\mathcal{N}_D$ and batchsize $\mathcal{N}_B$. As the Q-learning algorithm is known to overestimate action values, we also initialize action-value function parameter θ_M and target network θ_T with random weights to perform double Q-learning. Then, throughout the inner loop of each episode, the observed state s_t is passed to DRQN network giving to Q-value $(Q(s,a)_t)$ tensor for each possible action a_t at next position. The action is selected using ϵ-greedy exploration policy, executed and reward r_t is received before stepping up to next state s_{t+1}. Following, the transition is stored in experience reply $\mathcal{D}$. At the end, we sample a minibatch of $\mathcal{N}_B$ transition from $\mathcal{D}$ according to rank-based prioritization (Schaul et al., 2016), and update new network parameter using gradient descent. As can be seen in the Algorithm 1, the actions by DRQN are the same as in DQN, expect for the fact that information about the hidden state is also incorporated into the agent's observation space. For mathematical convenience, we have not used the hidden state notation in the algorithms. At the beginning of the episode, the hidden state is set to zero. The policy output is dependent on the hidden state (as described in Figure 2), as is the Q-function. The stacked LSTM layer maps the corresponding state, action and hidden sequence to new Q-values, as discussed in the algorithm.

To initialise the hidden state for sampling transitions while updating the DRQN network, we modify and use the method proposed by Kapturowski et al. (2019). We store the hidden recurrent states in experience replay and use it to initialise them periodically. The mini- batch is then sampled using rank-based prioritisation, as discussed in connection with Algorithm 1. The seminal DRQN paper by Hausknecht and Stone (2015) suggests two strategies: zero-initialising of the hidden state at the beginning of the sampled transition; and initialisation of whole transition. However, for both approaches, this "creates a number of practical, computational, and algorithmic issues due to varying and potentially environment-dependent sequence length, and higher variance of network updates because of the highly correlated nature of states in a trajectory when compared to training on randomly sampled batches of experience tuples" (Kapturowski et al., 2019).

In reinforcement learning, the agents inevitably confront a trade-off between exploration and exploitation (Ning et al., 2021). Exploration allows the agents to venture unsampled state space in search of larger reward. They can then employ all possible actions to improve Q-value estimates, thereby improving the action policy. By exploiting previously acquired knowledge, the agents exploit already learned Q-values to select the action with the greatest Q-value in the given state. We also implement an effective trade-off method: ϵ-greedy method. The ϵ-greedy exploration policy selects the action with the current greatest estimated Q-value with probability $1-\epsilon$ (exploitation) and randomly selects one action with

Algorithm 1 Deep Recurrent Q-Learning for High-Frequency Market Making

Initialize experience replay $\mathcal{D}$ of size $\mathcal{N}_D$ Initialize batch-size $\mathcal{N}_B$ Initialize action-value function parameter θ_M with random weights θ Initialize target network with weights $\theta_T = \theta$ Initialize ϵ, γ, N, ν

for *trading episode* $= 1$ *to* N **do**

  for *1* **to** T **do**

    Observes current state s_t Selects an action a_t with probability ϵ Otherwise randomly selects action $a_t = \arg\max_a Q(s_t, a; \theta)$ with probability $1 - \epsilon$ Execute the selected action a_t, get the reward r_t and next state s_{t+1} Store transition $(s_t, a_t, r_t, \gamma_t, s_{t+1})$ in experience replay $\mathcal{D}$ **for** $j \leftarrow 1$ **to** J **do**

      Sample a minibatch of $\mathcal{N}_B$ transition $\left(s_j, a_j, r_j, s_{j+1}\right)$ from $\mathcal{D}$ according to rank-based prioritization (Schaul et al., 2016) **if** $t_j = T$ **then**

        $$y_j(\theta_T) = r_j$$

      else

        $$y_j(\theta_T) = r_{(j)} + \gamma\, Q\left(s_j, a_j^* \middle| \theta_T\right)$$

      end

      where,

      $$a_j^* = \arg\max_a Q\left(s_{j-1}, a_{j-1} \mid \theta_M\right)$$

    end

    Obtain new network parameters θ_M by minimizing

    $$L(\theta; \theta_T) = \sum_{j=1}^{J} \left[y_j(\theta_T) - Q\left(s_j, a_j \mid \theta\right)\right]^2$$

    Using gradient descent to obtain $\theta_M = \arg\min_\theta L(\theta; \theta_T)$

  end

  Update target network $\theta_M = \theta_T$ after every ν iteration

end

probability ϵ (exploration) at each event time step. The value of ϵ decreases as training epochs progress, network estimate of Q is more accurate with training, and agents tend to exploit more often. We define decrease in ϵ by $\epsilon = \max\left(0.01, 1 - \frac{n}{N}\right)$. Here, n is the current training epoch and N is the total number of training epochs.

4. Experiments and Results

In this section, we first introduce the experimental setup, then define evaluation metrics in order to assess the performance of the market making strategy. We validate the model by reproducing known stylised facts from the simulated data, so that the dynamics of the simulated LOB's attributes match those observed empirically.

4.1. Experiments

We run the model for 1,000 iterations to find relevant hyper-parameter using random search. After that, we train the models for around ten million time steps, in intervals of 10,000, which is equivalent to 500 trading days of collecting data, monitoring and visualising the agent's learning. The environment is tested against the benchmark to determine the agent's learning pattern. In testing the network architecture, we use the parameters/hyperparameters described in Table 1. We have trained the deep reinforcement learning agent's policies using three different random seeds for each simulation. When performing the simulation, each high-frequency market making agent using different network architecture is trained against various high-frequency market taker agents until the point of convergence, then the final policy is used to report profit and loss.

Table 1: Selected Parameters.

Parameter/Hyperparameter	Value
Training episodes	500 days
Training sample size	400 days
Testing sample size	100 days
High-Frequency Market Makers initial capital	10000
High-Frequency Market Takers initial capital	10000
Min inventory ($\min \mathrm{Inv}$)	-20000
Max inventory ($\max \mathrm{Inv}$)	20000
Adverse selection (Υ)	0.1
Adverse selection by volatility (ξ)	0.05
Cancellation sensitivity (ψ)	0.01
Exponential penalty (α)	0.01
Experience replay ($\mathcal{D}$)	10^6
Batchsize ($\mathcal{D}$)	32
Target network update frequency (ν)	1000
DO nothing action (no-op max)	64
Optimizer learning rate (Adam)	10^{-4}

4.2. Performance Metrics

In our evaluation of the agents' performance, we use profit and loss with exponential transaction costs and maker-taker fees (PnL), computed for each hour. The trading strategy's efficiency in capturing the spread is evaluated by normalised daily PnL (NPnL) (Spooner et al., 2018). We also use the mean absolute position (MAP) to capture the important characteristics of market makers when agents avoid large inventories (Spooner et al., 2018). We report the NPnL and MAP with the standard deviation and mean, respectively. We use the spread-based benchmark strategy proposed by Spooner et al. (2018), which employs temporal-difference reinforcement learning to design market making agents. Here, the agents interacted with a data-driven simulation of a limit order book, at a depth of five

levels. The authors experimented with different kinds of agents to ensure that they chose the best consolidated agent. Their agents use an asymmetrically dampened reward function with a linear combination of tile codings, trained using SARSA. We adapted these agents to work with our simulation framework. As Table 2 shows, the agents' performance is below par. One possible reason for this is the complexity of the environment. While interacting with the benchmark's simulator, it is possible that an agent's trading strategies might follow historical trends and be executed at current market conditions, thus delineating them from market impact. However, our simulation framework is built over realistic market design and a matching engine that incorporates a dynamic transition fee, maker-taker fees, etc. As the DQN network is the foundation of deep reinforcement learning, we use the classic architecture. The network is trained with a Q-learning algorithm, stochastic gradient descent to update the weights and experience reply to alleviate the problems of correlated data and non-stationary distributions. Table 2 describes the performance of the DQN network.

Table 2: Normalised PnL (PnL) and mean absolute positions (MAP) for various agent architectures

Architecture	NPnL [10^5]		MAP[unit]	
	Mean	Std.Dev.	Mean	Std.Dev.
DRQN	1.4	±21.46	12	±16
DQN	0.2	±5.11	5	±7
RL	-1.2	±87.65	41	±72

4.3. Results and Analysis

The agents' performance is compared in Figure 3. Despite the handcrafted strategy, in which actions with various quantities are taken at different states, the RL agent performs badly, and is less stable than the DRQN and DQN agents. It is notable that the trading strategy followed by the RL agents doesn't take account of order size, cancellations, adverse selection, transaction costs and volatility, which the current simulator introduces into the interaction. In addition, the order matching is subject to market takers, who trade on market trends, as described in agent's trading strategies. The DQN's performance is stable, but fails to outperform the DRQN. This may be due to a lack of efficiency in state representation, overestimated action values, partial observability and pritorised experience, which DRQN incorporates. In addition, the agents designed on the basis of temporal-difference RL and DQN architecture have a restricted ability to take account of past observations in partially observable environments, and use these to implement efficient market making strategies. This is also reflected in Figure 3 and Table 2. The agents based on modified DRQN architecture use stacked LSTM layers that store temporal trading information and can learn sequences as they evolve over time. To better understand the performance, we need action selection that takes account of limit order book dynamics, which we plan to do next.

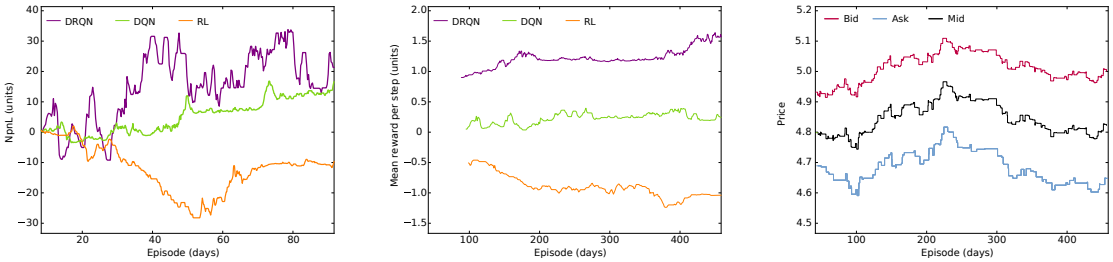

Figure 3: Agents' performance while training, testing and bid-ask distribution on a random day (left to right)

4.4. Impact of Maker-Taker Fees

The introduction of maker-taker fee structures in the equities market gives rise to a complex topology among high-frequency market making strategies and how they interact with other strategies. It is not clear whether the high-frequency market making based on the maker-taker fee model may be detrimental or beneficial to market quality. We would like to test the simulation framework's potential as a tool for improving our understanding of market design issues related to electronic exchanges. We investigate the effect of maker-taker fees on the agent's profitability and market quality with a deep agent-based model of the financial market. We use traditional bid-ask spread as an indicator of the market's liquidity and intraday realised volatility as a volatility measure – i.e. as a proxy for market quality. We increase the base rebate of providing liquidity (0.002) and taking liquidity (0.003) by 0.002 five times, to study the effect of maker-taker fees on market quality. The simulations were performed for each rebate value to report average profitability, daily realised volatility, and bid-ask spread over 500 simulation days. Figure 4 depicts market quality and agents' profitability for different rebate values.

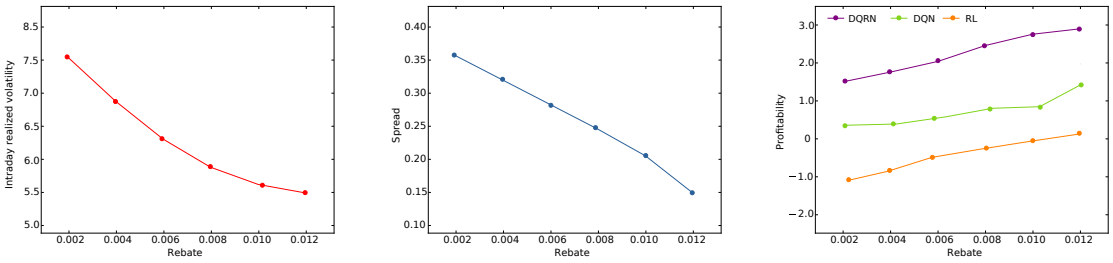

Figure 4: Market quality as maker-taker fees increase. Volatility, Liquidity and Agents' Profitability (left to right)

When the maker-taker rebate increases, the high-frequency market makers place an order at a tighter spread to make a profit from bid-ask spread and rebate, subject to the order execution. The tighter spread guarantees the order execution against taker strategies,

which in turn decreases the volatility (as shown in Figure 4, first) and decreases the bid-ask spread, thereby increasing the liquidity (as shown in Figure 4, second). The plunge in volatility, combined with the surge in liquidity, increases the high frequency market maker's profitability. Figure 4, third, substantiate the earlier claim. The high-frequency market makers add a stream of limit sell orders in front of the upward market trend to accumulate rebate plus profit as their orders get executed, which increases profitability. As shown in Figure 4, third, the RL agents bring their profit and loss close to zero taking advantage of the maker-taker pricing model.

4.5. Validation

In agent-based models of financial markets, it is standard practice to measure the validity of the model by investigating whether the order- book data have particular characteristics, known as "stylised facts" (Abergel et al., 2016). We present some of the stylized facts reproduced from historical order book data, as shown in Figure 5. To reproduce stylised facts, we first calculate the returns, which are given by $r(t) = \log(p_t) - \log(p_{t-1})$. The distribution of returns often comes with heavy tails (HT), as depicted in Figure 5, first. The normalised return distribution has a fatter tail than the green Gaussian distribution. Furthermore, the cumulative distributions function (Cont, 2001; Abergel et al., 2016), shown in green (positive tail) and yellow (negative tails) in Figure 5, second, exhibits power law (PL). The violet line represents the asymptotic power-law function with tail exponent 4. The absence of the autocorrelation (AAC) of price changes can be seen in Figure 5, third.

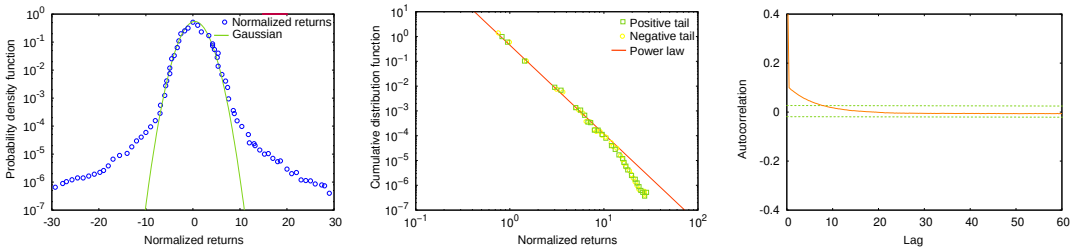

Figure 5: Stylised facts of simulated security returns. HT, PL and AAC (left to right)

5. Conclusions

In this paper, we have designed a market making agent using a deep recurrent Q-network that outperforms a prominent benchmark strategy based on temporal-difference reinforcement learning. The market making agents interacted with a highly realistic simulation of the limit order book, which has not previously been a subject of academic research. The suitable modification of DRQN's exciting network architecture (Hausknecht and Stone, 2015) and training procedure allowed our agents to achieve a predominant performance. We also investigated how the market maker's profitability increases with an increase in maker-taker fees, which in turn improves market quality. This paves the way for future research, e.g.: Refining the simulation framework to include latency in the agent's strategy. Extending to a portfolio with suitable hedging strategies rather than a single asset. Extending it to a

multi-agent setting in which all agents learn, compete and trade simultaneously. Incorporating order book data with deep reinforcement learning. Investigating how maker-taker fees affects market quality, the agents' interaction, and exchange competition in a fragmented market .

References

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

Jacob Abernethy and Satyen Kale. Adaptive market making via online learning. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2058–2066. Curran Associates, Inc., 2013.

Marco Avellaneda and Sasha Stoikov. High-frequency trading in a limit order book. *Quantitative Finance*, 8(3):217–224, 2008.

M. Bartolozzi. A multi agent model for the limit order book dynamics. *The European Physical Journal B*, 78(2):265–273, Nov 2010.

Aseem Brahma, Mithun Chakraborty, Sanmay Das, Allen Lavoie, and Malik Magdon-Ismail. A bayesian market maker. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, pages 215–232, 2012.

Álvaro Cartea, Sebastian Jaimungal, and Jason Ricci. Buy low, sell high: A high frequency trading perspective. *SIAM Journal on Financial Mathematics*, 5(1):415–444, 2014.

Tanmoy Chakraborty and Michael Kearns. Market making and mean reversion. In *Proceedings of the 12th ACM Conference on Electronic Commerce*, EC '11, pages 307–314, 2011. ISBN 978-1-4503-0261-6.

R. Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1(2):223–236, 2001.

Michael Dempster and V. Leemans. An automated fx trading system using adaptive reinforcement learning. *Expert Systems with Applications*, 30:543–552, 04 2006.

Jaimy van Dijk. Recurrent neural networks for reinforcement learning: an investigation of relevant design choices. Master's thesis, University of Amsterdam, Amsterdam, July 2017.

Marcus Elwin. Simulating market maker behavior using deep reinforcement learning to understand market microstructure. Master's thesis, KTH Royal Institute of Technology, Stockholm, January 2019.

Carole Comerton Forde and Tālis J. Putniņš. Dark trading and price discovery. *Journal of Financial Economics*, 118(1):70 – 92, 2015.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2094–2100. AAAI Press, 2016.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15)*, November 2015.

Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019.

Geir-Are Karvik, Joseph Noss, Jack Worlidge, and Daniel Beale. The deeds of speed: an agent-based model of market liquidity and flash episodes. Bank of England working papers 743, Bank of England, Jul 2018.

Frank McGroarty, Ash Booth, Enrico Gerding, and V.L.R Chinthalapati. High frequency trading strategies, market fragility and price spikes: an agent based model perspective. *Annals of Operations Research*, 282:217–244, 2019.

Albert Menkveld. High frequency trading and the new market makers. *Journal of Financial Markets*, 16(4):712–740, 2013.

Albert Menkveld. The economics of high-frequency trading: Taking stock. *Annual Review of Financial Economics*, 8(1):1–24, 2016.

Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 673–680, 2006. ISBN 1-59593-383-2.

Brian Ning, Franco Ho Ting Ling, and Sebastian Jaimungal. Double deep q-learning for optimal execution. *Applied Mathematical Finance*, 28(4):361–380, 2021.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. Market making via reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pages 434–442, Richland, SC, 2018.

P. Weber and B. Rosenow. Order book approach to price impact. *Quantitative Finance*, 5 (4):357–364, 2005.

Haoran Wei, Yuanbo Wang, Lidia Mangu, and Keith Decker. Model-based reinforcement learning for predictions and control for limit order books. *ArXiv*, arXiv:1910.03743, 2019.