
ASYNCHRONOUS DEEP DOUBLE DUELLING Q-LEARNING FOR TRADING-SIGNAL EXECUTION IN LIMIT ORDER BOOK MARKETS

Peer Nagy

Oxford-Man Institute of Quantitative Finance
University of Oxford
peer.nagy@eng.ox.ac.uk

Jan-Peter Calliess

Oxford-Man Institute of Quantitative Finance
University of Oxford

Stefan Zohren

Oxford-Man Institute of Quantitative Finance
University of Oxford

January 23, 2023

ABSTRACT

We employ deep reinforcement learning (RL) to train an agent to successfully translate a high-frequency trading signal into a trading strategy that places individual limit orders. Based on the ABIDES limit order book simulator, we build a reinforcement learning OpenAI gym environment and utilise it to simulate a realistic trading environment for NASDAQ equities based on historic order book messages. To train a trading agent that learns to maximise its trading return in this environment, we use Deep Duelling Double Q-learning with the APEX (asynchronous prioritised experience replay) architecture. The agent observes the current limit order book state, its recent history, and a short-term directional forecast. To investigate the performance of RL for adaptive trading independently from a concrete forecasting algorithm, we study the performance of our approach utilising synthetic alpha signals obtained by perturbing forward-looking returns with varying levels of noise. Here, we find that the RL agent learns an effective trading strategy for inventory management and order placing that outperforms a heuristic benchmark trading strategy having access to the same signal.

Keywords Limit Order Books · Quantitative Finance · Reinforcement Learning · LOBSTER ·

1 Introduction

Successful quantitative trading strategies often work by generating trading signals, which exhibit a statistically significant correlation with future prices. These signals are then turned into actions, aiming to assume positions in order to gain from future price changes. The higher the signal frequency and strategy turnover, the more critical is the execution component of the strategy, which translates the signal into concrete orders that can be submitted to a market. Such markets are oftentimes organised as an order ledger represented by a *limit order book (LOB)*.

Limit order book prices have been shown to be predictable over short time periods, predicting a few successive ticks into the future with some accuracy. This has been done by either utilising the recent history of order book states [1, 2], order-flow data [3], or market-by-order (MBO) data directly as features [4]. However, given the short time horizons over which these predictions are performed, and correspondingly small price movements, predictability does not directly translate into trading profits. Transaction costs, strategy implementation details, and time delays add up to the challenging problem of translating high-frequency forecasts into a trading strategy, which determines when and which orders to send to the exchange. Moreover, different predictive signals have to be traded differently to achieve optimal results, depending on the forecast horizon, signal stability, and predictive power.

In this paper, we use asynchronous off-policy reinforcement learning (RL), specifically Deep Duelling Double Q-learning with the APEX architecture [5], to learn an optimal trading strategy, given a noisy directional signal of short-term forward mid-quote returns. For this purpose, we developed an OpenAI gym [6] limit order book environment based on the ABIDES [7] market simulator, similar to [8]. We use this simulator to replay NASDAQ price-time priority limit order book markets using message data from the LOBSTER data set [9].

We study the case of an artificial or synthetic signal, taking the future price as known and adding varying levels of noise, allowing us to investigate learning performance and to quantify the benefit of an RL-derived trading policy compared to a baseline strategy using the same noisy signal. This is not an unrealistic setup when choosing the correct level of noise. Practitioners often have dedicated teams researching and deriving alpha signals, often over many years, while other teams might work on translating those signals into profitable strategies. Our aim is to focus on the latter problem which becomes increasingly more difficult as signals become faster. It is thus interesting to see how an RL framework can be used to solve this problem. In particular, we show that the RL agent learns superior policies to the baselines, both in terms of strategy return and Sharpe ratio. Machine learning methods, such as RL, have become increasingly important to automate trade execution in the financial industry in recent years [10], underlining the practical use of research in this area.

We make a number of contributions to the existing literature. By defining a novel action and state space in a LOB trading environment, we allow for the placement of limit orders at different prices. This allows the agent to learn a concrete high-frequency trading strategy for a given signal, trading either aggressively by crossing the spread, or conservatively, implicitly trading off execution probability and cost. In addition to the timing and level placement of limit orders, our RL agent also learns to use limit orders of single units of stock to manage its inventory as it holds variably sized long or short positions over time. More broadly, we demonstrate the practical use case of RL to translate predictive signals into limit order trading strategies, which is still usually a hand-crafted component of a trading system. We thus show that simulating limit order book markets and using RL to further automate the investment process is a promising direction for further research. To the best of our knowledge, this is also the first study applying the APEX [5] algorithm to limit order book environments.

The remaining paper is structured as follows: Section 2 surveys related literature, section 3 explains the mechanics of limit order book markets and the APEX algorithm, section 4 details the construction of the artificial price signal, section 5 showcases our empirical results, and section 6 concludes our findings.

2 Related Work

Reinforcement learning has been applied to learn different tasks in limit order book market environments, such as optimal trade execution [11, 12, 13, 14, 15], market making [16, 17], portfolio optimisation [18] or trading [19, 20, 21]. The objective of optimal trade execution is to minimise the cost of trading a predetermined amount of shares over a given time frame. Trading direction and the number of shares is already pre-defined in the execution problem. Market makers, on the other hand, place limit orders on both sides of the order book and set out to maximise profits from capturing the spread, while minimising the risk of inventory accumulation and adverse selection. We summarise using the term “RL for trading” such tasks which maximise profit from taking directional bets in the market. This is a hard problem for RL to solve as the space of potential trading strategies is large, leading to potentially many local optima in the loss landscape, and actionable directional market forecasts are notoriously difficult due to arbitrage in the market.

The work of [19] is an early study of RL for market microstructure tasks, including trade execution and predicting price movements. While the authors achieve some predictive power of directional price moves, forecasts are determined to be too erroneous for profitable trading. The most similar work to ours is [21] that provides the first end-to-end DRL framework for high-frequency trading, using PPO [22] to trade Intel stock. To model price impact, [21] use an approximation, moving prices proportionately to the square-root of traded volume. The action space is essentially limited to market orders, so there is no decision made on limit prices. The trained policy is able to produce a profitable trading strategy on the evaluated 20 test days. However, this is not compared to baseline strategies and the resulting performance is not statistically tested for significance. In contrast, we consider a larger action space, allowing for the placement of limit orders at different prices, thereby potentially lowering transaction costs of the learned HFT strategy. For a broader survey of deep RL (DRL) for trading, including portfolio optimisation, model-based and hierarchical RL approaches the reader is referred to [17].

3 Background

3.1 Limit Order Book Data

Limit order books (LOBs) are one of the most popular financial market mechanisms used by exchanges around the world [23]. Market participants submit limit buy or sell orders, specifying a maximum (minimum) price at which they are willing to buy (sell), and the size of the order. The exchange’s limit order book then keeps track of unfilled limit orders on the buy side (bids) and the sell side (asks). If an incoming order is marketable, i.e. there are open orders on the opposing side of the order book at acceptable prices, the order is matched immediately, thereby removing liquidity from the book. The most popular matching prioritisation scheme is price-time priority. Here, limit orders are matched first based on price, starting with the most favourable price for the incoming order, and then based on arrival time, starting with the oldest resting limit order in the book, at each price level. For a more complete review of limit order book dynamics and pertaining models, we refer the reader to [23].

In this paper, we consider equity limit order book data from the NASDAQ exchange [9], which also uses a price-time priority prioritisation. Our market simulator keeps track of the state of the LOB by replaying historical message data, consisting of new incoming limit orders, order cancellations or modifications. The RL agent can then inject new messages into the order flow and thereby, change the LOB state from its observed historical state.

Our simulator reconstructs LOB dynamics from message data, so every marketable order takes liquidity from the book and thus has a direct price impact. Beyond that, we make no further assumptions on permanent market impact or reactions of other agents in the market, which we leave to future work.

3.2 Double DQN with Distributed Experience Replay

We model the trader’s problem as a Markov Decision Process (MDP) [24, 25], described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$. \mathcal{S} denotes a state space, \mathcal{A} an action space, \mathcal{T} a stochastic transition function, r a reward function and γ a discount factor. Observing the current environment state $s_t \in \mathcal{S}$ at time t , the trader takes action $a_t \in \mathcal{A}$, which causes the environment to transition state according to the stochastic transition function $\mathcal{T}(s_{t+1}|s_t, a_t)$. After transitioning from s_t to s_{t+1} , the agent receives a reward $r_{t+1} = r(s_t, a_t, s_{t+1})$. We use Deep Double Q-learning [26] with a duelling network architecture [27] to approximate the optimal Q-function $Q^*(s, a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | a_t = a, s_t = s]$. To speed up the learning process we employ the *APEX* training architecture [5], which combines asynchronous experience sampling using parallel environments with off-policy learning from experience replay buffers. Every episode i results in an experience trajectory $\tau_i = \{s_t, a_t\}_{t=1}^T$, many of which are sampled from parallel environment instances and are then stored in the replay buffer. The environment sampling is done asynchronously using parallel processes running on CPUs. Experience data from the buffer is then sampled randomly and batched to perform a policy improvement step of the Q-network on the GPU. Prioritised sampling from the experience buffer has proven to degrade performance in our noisy problem setting, hence we are sampling uniformly from the buffer.¹ After a sufficient number of training steps, the new policy is then copied to every CPU worker to update the behavioural policy.

Double Q-learning [28, 26] stabilises the learning process by keeping separate Q-network weights for action selection (main network) and action validation (target network). The target network weights are then updated gradually in the direction of the main network’s weight every few iterations. The duelling network architecture [27] on the other hand uses two separate network branches (for both main and target Q-networks). One branch estimates the value function $V(s) = \max_a Q(s, a)$, while the other estimates the advantage function $A(s, a) = Q(s, a) - V(s)$. The benefit of this architecture choice lies therein that the advantage of individual actions in some states might be irrelevant, and the state value, which can be learnt more easily, suffices for an action-value approximation.

4 Framework

4.1 Artificial Price Signal

The artificial directional price signal $d_t \in \Delta^2 = \{x \in \mathbb{R}^3 : x_1 + x_2 + x_3 = 1, x_i \geq 0 \text{ for } i = 1, 2, 3\}$ the agent receives is modelled as a discrete probability distribution over 3 classes, corresponding to the averaged mid-quote price decreasing, remaining stable, or increasing over a fixed future time horizon of $h \in \mathbb{N}_+$ seconds. To achieve realistic levels of temporal stability of the signal process, d_t is an exponentially weighted average, with persistence coefficient

¹In many application domains prioritised sampling, whereby we resample instances more frequently where the model initially performs poorly tends to aide learning. However, in our low signal-to-noise application domain, we noted poor performance. Investigating the matter, we found that prioritised sampling caused more frequent resampling of highly noisy instances where learning was particularly difficult, hence degrading performance.

$\phi \in (0, 1)$, of Dirichlet random variables ϵ_t . The Dirichlet parameters α depend on the realised smoothed future return $\overline{r_{t+h}}$, specifically on whether the return lies within a neighbourhood of size k around zero, or above or below. Thus we have:

$$\begin{aligned} d_t &= \phi d_{t-1} + (1 - \phi) \epsilon_t \\ \epsilon_t &= \text{Dirichlet}(\alpha(\overline{r_{t+h}})) \\ \overline{r_{t+h}} &= \frac{\overline{p_{t+h}} - p_t}{p_t} \quad \text{where} \quad \overline{p_{t+h}} = \frac{1}{h} \sum_{i=1}^h p_{t+i} \end{aligned} \quad (1)$$

and prices p_t refer to the mid-quote price at time t . The Dirichlet distribution is parametrised, so that, in expectation, the signal d_t updates in the direction of future returns, where a^H and a^L determine the variance of the signal. The Dirichlet parameter vector is thus:

$$\alpha(r_{t+h}) = \begin{cases} (a^H, a^L, a^L) & \text{if } \overline{r_{t+h}} < -k \\ (a^L, a^H, a^L) & \text{if } -k \leq \overline{r_{t+h}} < k \\ (a^L, a^L, a^H) & \text{if } k \leq \overline{r_{t+h}}. \end{cases} \quad (2)$$

4.2 RL Problem Specification

At each time step t , the agent receives a new state observation s_t . s_t consists of the time left in the current episode $T - t$ given the episode's duration of T , the agent's cash balance C_t , stock inventory X_t , the directional signal $d_t \in \Delta^2$, encoded as probabilities of prices decreasing, remaining approximately constant, or increasing; and price and volume quantities for the best bid and ask (level 1), including the agent's own volume posted at bid and ask: $o_{b,t}$ and $o_{a,t}$ respectively. In addition to the most current observable variables at time t , the agent also observes a history of the previous l values, which are updated whenever there is an observed change in the LOB. Putting all this together, we obtain the following state observation:

$$s_t = \left(\begin{array}{c} T - u \\ C_u \\ X_u \\ (d_u^1, d_u^2, d_u^3)' \\ (p_{a,u}, v_{a,u}, o_{a,u}, p_{b,u}, v_{b,u}, o_{b,u})' \end{array} \right)_{u=\{t-l, \dots, t\}}.$$

After receiving the state observation, the agent then chooses an action a_t . It can place a buy or sell limit order of a single share at bid, mid-quote, or ask price; or do nothing and advance to the next time step. Actions, which would immediately result in positions outside the allowed inventory constraints $[pos_{min}, pos_{max}]$ are disallowed and do not trigger an order. Whenever the execution of a resting limit order takes the inventory outside the allowed constraints, a market order in the opposing direction is triggered to reduce the position back to pos_{min} for short positions or pos_{max} for long positions. Hence, we define

$$a_t \in \mathcal{A} = (\{-1, 1\} \times \{-1, 0, 1\}) \cup \{skip\}$$

so that in total there are 7 discrete actions available, three levels for both buy and sell orders, and a skip action. For the six actions besides the “skip” action, the first dimension encodes the trading direction (sell or buy) and the second dimension the price level (bid, mid-price, or ask). For example, $a = (1, 0)$ describes the action to place a buy order at the mid price, and $a = (-1, 1)$ a sell order at best ask. Rewards R_{t+1} consist of a convex combination of a profit-and-loss-based reward R_{t+1}^{pnl} and a directional reward R_{t+1}^{dir} . R_{t+1}^{pnl} is the log return of the agent's mark-to-market portfolio value M_t , encompassing cash and the current inventory value, marked at the mid-price. The benefit of log-returns is that they are additive over time, rather than multiplicative like gross returns, so that, without discounting ($\gamma = 1$) the total profit-and-loss return $\sum_{s=t+1}^T R_s^{pnl} = M_T - M_t$. The directional reward term R_{t+1}^{dir} incentivizes the agent to hold inventory in the direction of the signal and penalises the agent for inventory positions opposing the signal direction. The size of the directional reward can be scaled by the parameter $\kappa > 0$. R_{t+1}^{dir} is positive if the positive prediction has a higher score than the negative ($d_{t,3} > d_{t,1}$) and the current inventory is positive; or if $d_{t,3} < d_{t,1}$ and

$X_t < 0$. Further, if the signal $[-1, 0, 1] \cdot d_t$ has an opposite sign than inventory X_t , R_{t+1}^{dir} is negative. This can be summarised as follows:

$$\begin{aligned}
\text{Mark-to-Market Value} \quad M_t &= C_t + X_t p_t^m \\
&\Delta M_t = \Delta C_t + X_{t-1} \Delta p_t^m + \Delta x_t p_t^m \\
\text{PnL Reward} \quad R_{t+1}^{pnl} &= \ln(M_t) - \ln(M_{t-1}) \\
\text{Directional Reward} \quad R_{t+1}^{dir} &= \kappa[-1, 0, 1] \cdot d_t X_t \\
\text{Total Reward} \quad r_{t+1} &= w^{dir} R_{t+1}^{dir} + (1 - w^{dir}) R_{t+1}^{pnl}
\end{aligned} \tag{3}$$

The weight on the directional reward $w^{dir} \in [0, 1)$ is reduced every learning step by a factor $\psi \in (0, 1)$,

$$w^{dir} \leftarrow \psi w^{dir}$$

so that initially the agent quickly learns to trade in the signal direction. Over the course of the learning process, R_t^{pnl} becomes dominant and the agent maximises its mark-to-market profits.

5 Experimental Results

We train all RL policies using the problem setup discussed in section 4.2 on 5.5 months of Apple (AAPL) limit order book data (2012-01-01 to 2012-05-16) and evaluate performance on 1.5 months of out-of-sample data (2012-05-17 to 2012-06-31). We only use the first hour of every trading day (09:30 to 10:30) as the opening hour exhibits higher than average trading volume and price moves. Each hour of the data corresponds to a single RL episode.

Our neural network architecture consists of 3 feed-forward layers, followed by an LSTM layer, for both the value- and advantage stream of the duelling architecture. The LSTM layer allows the agent to efficiently learn a memory-based policy with observations including 100 LOB states.

We compare the resulting learned RL policies to a baseline trading algorithm, which receives the same artificially perturbed high-frequency signal of future mid-prices. The baseline policy trades aggressively by crossing the spread whenever the signal indicates a directional price move up or down until the inventory constraint is reached. The signal direction in the baseline algorithm is determined as the prediction class with the highest score (down, neutral, or up). When the signal changes from up or down to neutral, indicating no immediate expected price move, the baseline strategy places a passive order to slowly reduce position size until the inventory is cleared. This heuristic utilises the same action space as the RL agent and yielded better performance than trading using only passive orders (placed at the near touch), or only aggressive orders (at the far touch).

Figure 1 plots a 17 second simulation window from the test period, comparing the simulated baseline strategy with the RL strategy. It can be seen that prices in the LOB are affected by the trading activity as both strategies inject new order flow into the market, in addition to the historical orders, thereby consuming or adding liquidity at the best bid and ask. During the plotted period, the baseline strategy incurs small losses due to the signal switching between predicting decreasing and increasing future prices. This causes the baseline strategy to trade aggressively, paying the spread with every trade. The RL strategy, on the other hand, navigates this difficult period better by trading more passively out of its long position, and again when building up a new position. Especially in the second half of the depicted time period, the RL strategy adds a large number of passive buy orders (green circles in the second panel of figure 1). This is shown by the green straight lines, which connect the orders to their execution or cancellation, some of which occur after the depicted period.

5.1 Oracle Signal

The RL agent receives a noisy oracle signal of the mean return $h = 10$ seconds into the future (see equation 4.1). It chooses an action every 0.1s, allowing a sufficiently quick build-up of long or short positions using repeated limit orders of single stocks. The algorithm is constrained to keep the stock inventory within bounds of $[pos_{min}, pos_{max}] = [-10, 10]$. To change the amount of noise in the signal, we vary the a^H parameter of the Dirichlet distribution, keeping $a^L = 1$ constant in all cases. To keep the notation simple, we hence drop the H superscript and refer to the variable Dirichlet parameter a^H simply as a . We consider three different noise levels, parametrising the Dirichlet distribution with $a = 1.6$ (low noise), $a = 1.3$ (mid noise), and $a = 1.1$ (high noise). A fixed return classification threshold $k = 4 \cdot 10^{-5}$ was chosen to achieve good performance of the baseline algorithm, placing around 85% of observations in the up or down category. The signal process persistence parameter is set to $\phi = 0.9$.

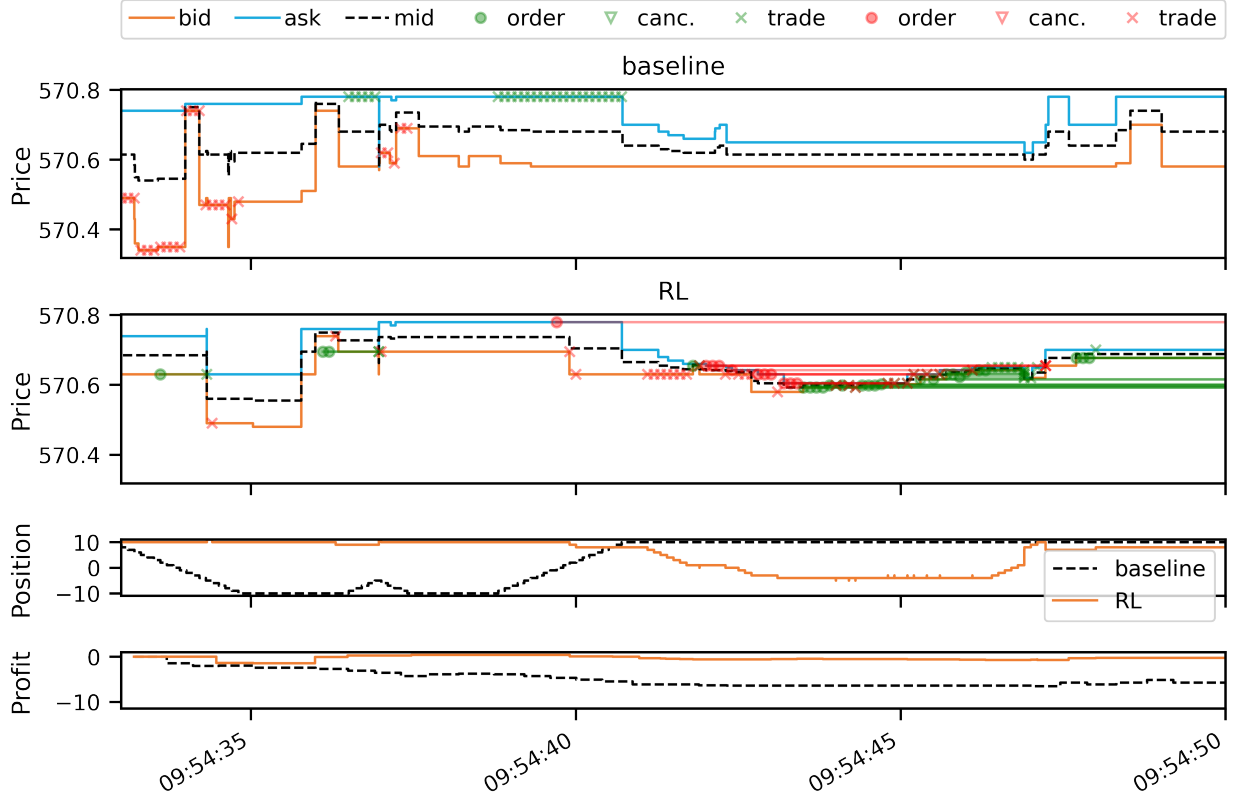


Figure 1: A short snapshot of simulation results (AAPL on 2012-06-14), comparing the RL policy (second panel) with the baseline (first panel). The first two panels plot the best bid, ask, and mid-price, overlaying trading events of buy orders (green) and sell orders (red). Circles mark new unmarketable limit orders entering the book. Crosses mark order executions (trades) and triangles order cancellations. Open orders are connected by lines to either cancellations or trades. Since we are simulating the entire LOB, trading activity can be seen to affect bid and ask prices. The third panel plots the evolution of the inventory position of both strategies, and the last panel the trading profits over the period in USD.

Out-of-sample trading performance is visualised by the account curves in figure 2. The curves show the evolution of the portfolio value for a chronological evaluation of all test episodes. Every account curve shows the mean episodic log-return μ and corresponding Sharpe ratio S next to it. We show that all RL-derived policies are able to outperform their respective baseline strategies for the three noise levels investigated. Over the 31 test episodes, the cumulative RL algorithm out-performance over the baseline strategy ranges between 14.8 ($a = 1.3$) and 32.2 ($a = 1.1$) percentage points (and 20.7 for $a = 1.6$). In the case of the signal with the lowest signal-to-noise ratio ($a=1.1$), for which the baseline strategy incurs a loss for the test period, the RL agent has learned a trading strategy with an approximately zero mean return. Temporarily, the strategy even produces positive gains. Overall, it produces a sufficiently strong performance to not lose money while still trading actively and incurring transaction costs. Compared to a buy-and-hold strategy over the same time period, the noisy RL strategy similarly produces temporary out-performance, with both account curves ending up flat with a return around zero. Inspecting Sharpe ratios, we find that using RL to optimise the trading strategy is able to increase Sharpe ratios significantly. The increase in returns of the RL strategies is hence not simply explained by taking on more market risk.

Figure 3a compares the mean return between the buy & hold, baseline, and RL policies for all out-of-sample episodes across the three noise levels. A single dashed grey line connects the return for a single test episode across the three trading strategies: buy & hold, baseline, and the RL policy. The solid blue lines representing the mean return across all episodes. Error bars represent the 95% bootstrapped confidence intervals for the means. Testing for the significance of the differences between RL and baseline returns across all episodes (t-test) is statistically significant ($p \ll 0.1$) for all noise levels. Differences in Sharpe ratios are similarly significant. We can thus conclude that the high frequency trading strategies learned by RL outperform our baseline strategy for all levels of noise we have considered.

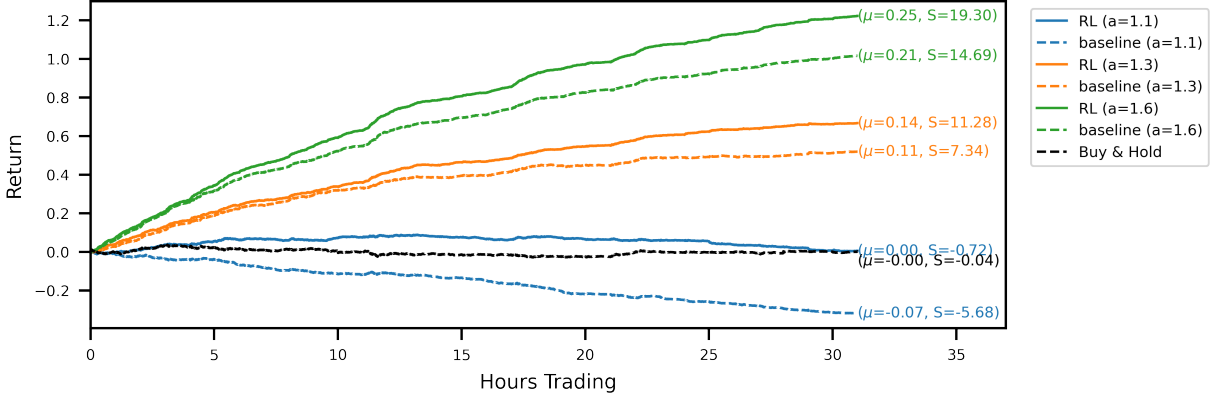


Figure 2: Account curves, trading the noisy oracle signal in the test set, comparing the learned RL policies (solid lines) with the baseline trading strategy (dashed). The black line shows the performance of the buy & hold strategy over the same period. Different colours correspond to different signal noise levels. The RL policy is able to improve the trading performance across all signal noise levels.

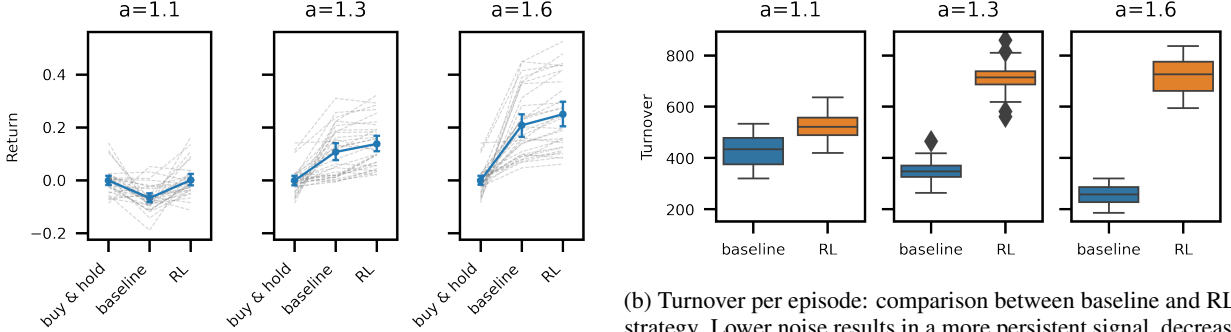


Figure 3: Mean return and turnover of the baseline and RL trading strategies.

It is also informative to compare the amount of trading activity between the baseline and RL strategies (see figure 3b). The baseline turnover decreases with an increasing signal-to-noise ratio (higher a), as the signal remains more stable over time, resulting in fewer trades. In contrast, the turnover of the RL trading agent increases with a higher signal-to-noise ratio, suggesting that the agent learns to trust the signal more and reflecting that higher transaction costs, resulting from the higher trading activity, can be sustained, given a higher quality signal. In the high noise case ($a = 1.1$), the RL agent learns to reduce trading activity relative to the other RL strategies, thereby essentially filtering the signal. The turnover is high in all cases due to the high frequency of the signal and the fact that we are only trading a small inventory. Nonetheless, performance is calculated net of spread-based transaction costs as our simulator adequately accounts for the execution of individual orders.

Table 1 lists action statistics for all RL policies, including how often actions are skipped, and the price levels at which limit orders are placed, grouped by buy and sell orders. With the least informative signal, the strategy almost exclusively uses marketable limit orders, with buy orders being placed at the bid and sell orders at the ask price. With better signals being available ($a = 1.3$ and $a = 1.6$), buy orders are more often placed at the mid-quote price, thereby trading less aggressively and saving on transaction costs. Overall, the strategies trained on different signals all place the majority of sell orders at the best bid price, with the amount of skipped actions varying considerably across the signals.

	a=1.1	a=1.3	a=1.6
action skipped [%]	24.5	43.8	7.8
sell levels (bid, mid, ask) [%]	(95.4, 3.1, 1.5)	(94.6, 2.8, 2.65)	(97.2, 1.9, 0.9)
buy levels (bid, mid, ask) [%]	(1.1, 1.3, 97.5)	(1.6, 52.9, 45.5)	(1.7, 13.0, 85.3)

Table 1: Actions taken by RL policy for the three different noise levels: the first row shows how often the policy chooses the “skip” action. Not choosing this action does however not necessarily result in an order being placed, e.g. if inventory constraints are binding. The last two rows show the relative proportion of limit order placement levels for sell orders, and buy orders, respectively.

6 Conclusions

Using Deep Double Duelling Q-learning with asynchronous experience replay, a state-of-the-art off-policy reinforcement learning algorithm, we train a limit order trading strategy in an environment using historic market-by-order (MBO) exchange message data. For this purpose we develop an RL environment based on the ABIDES [7] market simulator, which reconstructs order book states dynamically from MBO data. Observing an artificial high-frequency signal of the mean return over the following 10 seconds, the RL policy successfully transforms a directional signal into a limit order trading strategy. The policies acquired by RL outperform our baseline trading algorithm, which places marketable limit orders to trade into positions and passive limit orders to exit positions, both in terms of mean return and Sharpe ratio. We investigate the effect of different levels of noise in the alpha signal on the RL performance. Unsurprisingly, more accurate signals lead to higher trading returns but we also find that RL provides a similar added benefit to trading performance across all noise levels investigated.

The task of converting high-frequency forecasts into tradeable and profitable strategies is difficult to solve as transaction costs, due to high portfolio turnover, can have a prohibitively large impact on the bottom line profits. We suggest that RL can be a useful tool to perform this translational role and learn optimal strategies for a specific signal and market combination. We have shown that tailoring strategies in this way can significantly improve performance, and eliminates the need for manually fine-tuning execution strategies for different markets and signals. For practical applications, multiple different signals could even be combined into a single observation space. That way the problem of integrating different forecasts into a single coherent trading strategy could be directly integrated into the RL problem.

While we here show an interesting use-case of RL in limit order book markets, we also want to motivate the need for further research in this area. There are many years of high-frequency market data available, which ought to be utilised to make further progress in LOB-based tasks and improve RL in noisy environments. This, together with the newest type of neural network architectures, such as attention-based transformers [29, 30], enables learning tasks in LOB environments directly from raw data with even better performance. For the task we have considered in this paper, future research could enlarge the action space, allowing for placement of limit orders deeper into the book and larger orders sizes. Allowing for larger sizes however would require a realistic model of market impact, considering the reaction of other market participants.

References

- [1] Zihao Zhang, Stefan Zohren, and Stephen Roberts. DeepLOB: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, 2019.
- [2] Zihao Zhang and Stefan Zohren. Multi-horizon forecasting for limit order books: Novel deep learning approaches and hardware acceleration using intelligent processing units. *arXiv preprint arXiv:2105.10430*, 2021.
- [3] Petter N Kolm, Jeremy Turiel, and Nicholas Westray. Deep order flow imbalance: Extracting alpha at multiple horizons from the limit order book. *Available at SSRN 3900141*, 2021.
- [4] Zihao Zhang, Bryan Lim, and Stefan Zohren. Deep learning for market by order data. *Applied Mathematical Finance*, 28(1):79–95, 2021.
- [5] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.
- [6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [7] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. Abides: Towards high-fidelity market simulation for AI research. *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 11–22, 2020.
- [8] Selim Amrouni, Aymeric Moulin, Jared Vann, Svitlana Vyetenko, Tucker Balch, and Manuela Veloso. ABIDES-gym: gym environments for multi-agent discrete event simulation and application to financial markets. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9. ACM, 2021.
- [9] Ruihong Huang and Tomas Polak. Lobster: Limit order book reconstruction system. *Available at SSRN 1977207*, 2011.
- [10] Peer Nagy, James Powrie, and Stefan Zohren. Machine learning for microstructure data-driven execution algorithms. In *The Handbook on AI and Big Data Applications in Investments*. CFA Institute Research Foundation, forthcoming 2023.
- [11] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680, 2006.
- [12] Brian Ning, Franco Ho Ting Lin, and Sebastian Jaimungal. Double deep Q-learning for optimal execution. *Applied Mathematical Finance*, 28(4):361–380, 2021.
- [13] Kevin Dabérius, Elvin Granat, and Patrik Karlsson. Deep execution-value and policy based reinforcement learning for trading and beating market benchmarks. *Available at SSRN 3374766*, 2019.
- [14] Michäel Karpe, Jin Fang, Zhongyao Ma, and Chen Wang. Multi-agent reinforcement learning in a realistic limit order book market simulation. In *Proceedings of the First ACM International Conference on AI in Finance*. ACM, 2020.
- [15] Matthias Schnaubelt. Deep reinforcement learning for the optimal placement of cryptocurrency limit orders. *European Journal of Operational Research*, 296(3):993–1006, 2022.
- [16] Jacob D Abernethy and Satyen Kale. Adaptive market making via online learning. In *Advances in Neural Information Processing Systems*, volume 26, 2013.
- [17] Pankaj Kumar. Deep reinforcement learning for market making. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1892–1894, 2020.
- [18] Pengqian Yu, Joon Sern Lee, Ilya Kulyatin, Zekun Shi, and Sakyasingha Dasgupta. Model-based deep reinforcement learning for dynamic portfolio optimization. *arXiv preprint arXiv:1901.08740*, 2019.
- [19] Michael Kearns and Yuriy Nevmyvaka. Machine learning for market microstructure and high frequency trading. *High Frequency Trading: New Realities for Traders, Markets, and Regulators*, 2013.
- [20] Haoran Wei, Yuanbo Wang, Lidia Mangu, and Keith Decker. Model-based reinforcement learning for predictions and control for limit order books. *arXiv preprint arXiv:1910.03743*, 2019.
- [21] Antonio Briola, Jeremy Turiel, Riccardo Marcaccioli, and Tomaso Aste. Deep reinforcement learning for active high frequency trading. *arXiv preprint arXiv:2101.07107*, 2021.
- [22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [23] Martin D Gould, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. Limit order books. *Quantitative Finance*, 13(11):1709–1742, 2013.

- [24] Richard Bellman. A Markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.
- [25] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.
- [26] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.
- [27] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003. PMLR, 2016.
- [28] Hado Hasselt. Double Q-learning. *Advances in Neural Information Processing Systems*, 23, 2010.
- [29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.
- [30] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- [31] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR, 2018.

7 Appendix

We use the RLLib library [31] for a reference implementation of the APEX algorithm. Table 2 shows a selection of relevant parameters we used for RL training.

Paramter	Value
timesteps_total	300e6
framework	torch
num_gpus	1
num_workers	42
batch_mode	truncate_episode
gamma	.99
lr_schedule	[[0,2e-5], [1e6, 5e-6]]
buffer_size	2e6
learning_starts	5000
train_batch_size	50
rollout_fragment_length	50
target_network_update_freq	5000
n_step	3
prioritized_replay	False

Table 2: Selected RL parameters for APEX algorithm using RLLib [31] library for training.

Figure 4 shows confusion matrices interpreting the oracle signal scores as probabilities over the three classes: down, stationary, and up. The predicted class is thus the one with the highest score.

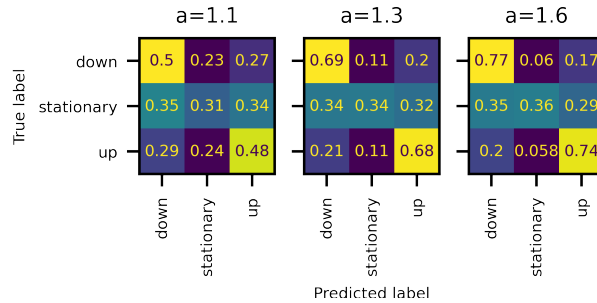


Figure 4: Confusion matrices of the artificial oracle signal for three noise levels, from low to high noise.