Reinforcement Learning for High Frequency Trading

***Abstract*—We produce a complete environment for high frequency trading and train an agent on it with deep Q-learning. Results show that the agent outperforms random decision making, highlighting its potential for real trading use.**

I. Introduction

Reinforcement learning has completely reshaped our way of dealing with complex problems. Indeed, it has shown positive results on several highly complicated modeling tasks, beating our current solvers by a great margin. Recently, deep reinforcement learning has demonstrated its capacity to solve problems with continuous spaces and decision states, highlighting its potential.

In this paper, we focus on deep Q-learning for high frequency trading on a simulated limit order book environment. We create a complete environment similar to a real market and train our agent on it. In particular, we investigate how a DQL agent can learn to execute trading decisions—such as market orders, limit orders, and cancellations—in a dynamic market simulation where order arrivals, cancellations, and additions are modeled via stochastic (Poisson) processes.

High frequency trading has gained rising popularity in recent years and is primarily a sequential decision-making problem under uncertainty, which makes it an ideal testbed for RL. The ability to learn effective trading strategies directly from interaction data could have significant practical implications in algorithmic trading, risk management, and automated execution. Moreover, the challenges posed by non-stationary market dynamics, sparse rewards, and high noise levels push the boundaries of current RL methodologies.

The inherently stochastic nature of financial markets makes RL a huge challenge, as our agent will need to adapt to every scenario. Moreover, it is not possible to train an AI agent on real data since we do not have the possibility to interact with the environment. It is thus necessary to design a complete environment to emulate markets and train our agent on it before deploying it in real markets. In order to simulate a complete limit order book (i.e., every event on the bid and ask side), we need a concrete modeling of a financial market, incorporating its key components such as market impact [1]. Specific models have been developed over the years, including the recent queue-reactive model [2]. This model has the capacity to recreate key components of real financial markets, such as price impact—i.e., how our decisions affect the market—and the well-known mid-price imbalance relationship. However, recent studies [3] have shown how the queue-reactive model can be adapted for a better understanding of the markets. We will thus create a new dynamic model based on these recent observations.

Previous studies have explored various RL methods—such as DQL, policy gradients, and actor-critic architectures—for financial trading [4] and [5]. While several environments and simulation frameworks are available (e.g., OpenAI Gym-based trading environments or specialized packages like RLTrader), many of these simplify the order-book dynamics or do not fully capture the complexities of real-world trading. Our work builds on this literature by incorporating a more detailed market simulation with explicit modeling of order additions, cancellations, and executions.

We designed a new simulation environment that mimics realistic market behavior by modeling the order book with multiple price levels and incorporating Poisson processes to generate events (orders, additions, cancellations). A Deep Q-Network (DQN) is then trained in this environment:

- **Environment Design:** The simulation code models the evolution of bid and ask queues and updates the market state based on stochastic events.
- **Agent Design:** A DQN-based trading agent is implemented using PyTorch. The network takes as input a state vector (price, time, bid/ask levels, and the agent's position) and outputs Q-values for a discrete set of trading actions.
- **Training Pipeline:** A replay buffer is used to stabilize learning, and the agent is trained over multiple episodes with a defined reward function based on net asset value changes.

Our experimental results indicate that the DQL agent can learn a trading policy that outperforms a random strategy in terms of cumulative profit. The agent gradually learns to balance between aggressive market orders and patient limit orders to optimize its trading performance. These findings suggest that even with a simplified market model, RL can capture key elements of optimal trading strategies.

However, it is important to highlight important improvements and limitation for our future work:

- **Market Realism:** The current simulation abstracts many aspects of real markets, such as latency and order matching complexities.
- **Algorithm Complexity:** While DQL provides a baseline, more advanced RL techniques (e.g., actor-critic methods, distributional RL) might yield better performance.
- **Scalability:** Extending the model to multi-agent settings or higher-frequency data remains a challenge. Future

work could focus on addressing these limitations by integrating more realistic market features and testing alternative RL architectures.

The complete anonymous code and implementation is provided (https://github.com/edlaf/Deep-Learning-for-High-Frequency-trading.git) to reproduce our results and to serve as a basis for further research.

II. Environment, Agent description and Deep Q-Learning

The electronification of financial markets at the end of the last century was a major turning point in market operations. The massive adoption of technology and electronic systems has gone hand in hand with an intensification in the number of trades carried out each day (see Figure), and has resulted in a drastic reduction in the execution time of market orders, which today have a frequency of around ten microseconds - creating a whole new structure on a microeconomic scale.

A. Environment and Agent

Interaction between buyers and sellers takes place algorithmically, according to the order of arrival of orders in the supply and demand queues. This process is managed by the Limit Order Book, which lists and executes all orders sent by the various market players. These modifications are of three main types:

- **Limit Order** : Add a buy or sell proposal at a given price.
- **Market Order** : Buy or sell immediately at the best price available on the market.
- **Cancellation** : Withdrawal of a buy or sell proposal previously registered in the order book.

The market microstructure created by these interactions between buyers and sellers is at the root of the price formation process, and drives price variations over time.

In a real market a trader have the possibility to make these actions on spectific prices. Indeed, one of the main characteristics of HF is that prices can only take on multiple values of a quantity called **tick**. The trader then has the possibility to do these 3 actions on the buy side and on the sell side when he possesses some stocks at any tick multiple (any possible price). The closest price of the actual price is called the first limit. After this first limit we have the second limit ne tick after the first, the third two ticks after the first, ect. Currently our agent only hase the possibility to interact with the first limit. It can thus decide to either buy at any time (market order buy side), add itself to the first limit (add) or cancel if has previously been added. We it posses a stock it has the possibility to do the same on the sell side. Currently, these actions have no prices, which is of course not realistic. We will incorporate these changes and the possibility to interact with other limits in futur work. We will also incorporate latency to simulate a more realistic market. Moreover, our agent is only capable of interacting with the market every 50 actions, has it will have too big of an impact otherwise. It also currently only hase the possibility to only add/cancel/buy/sell for 1-size orders (add 1, buy 1,...). Finally it also has the possibilty to do nothing.

The action state is thus defined by these six actions (or 3 if no asset is currently hold). The state is defined has the current price and the 3 first limit on the bid and ask side. It is thus a 7-dimensional vector of $\mathbb{N}_+^6 \times \mathbb{R}$.:

$$s_t \in \mathbb{N}_+^6 \times \mathbb{R}$$

$$a_t \in \{\text{Do nothing, Market order, Add, Cancel}\} \times \{\text{Bid, Ask}\}$$

With these different action, we train our agent with classic deep Q-learning.

B. Deep Q-Learning Application

The general principle of Deep Q-Learning is to approximate the action-value function (or *Q-function*) using a deep neural network:

$$Q(s, a; \theta) \approx \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k \, r_{t+k} \,\Big|\, s_t = s,\, a_t = a\Big],$$

where θ represents the network's parameters (weights), s is a state, a is an action, r_{t+k} are future rewards, and $\gamma \in [0, 1]$ is the discount factor.

a) Temporal-Difference (TD) Learning: The neural network is trained to predict the following *Bellman target*:

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-),$$

where θ^- denotes the parameters of the *target network*, which remain fixed for a certain number of iterations to stabilize the learning process. The weights θ are updated via back-propagation, by minimizing the mean-squared error between $Q(s_t, a_t; \theta)$ and y_t:

$$\mathcal{L}(\theta) \;=\; \mathbb{E}\big[\big(Q(s_t, a_t; \theta) - y_t\big)^2\big].$$

b) Experience Replay: To break the strong correlation between consecutive samples and make learning more stable, the transitions (s_t, a_t, r_t, s_{t+1}) are stored in a *replay buffer*. The agent then samples mini-batches of transitions randomly from this buffer to train the network.

c) Exploration: In Deep Q-Learning, the agent continually faces a trade-off between exploiting its current knowledge and exploring new actions that could lead to higher returns. Even though the ultimate goal is to take the best possible action in every state (i.e., exploit the policy), relying solely on exploitation from the outset risks overlooking better strategies that are discovered later. Consequently, an exploration mechanism such as ϵ-greedy is employed. With probability ϵ, the agent takes a random action, irrespective of the current Q-value estimate. By doing so, it can gather diverse experiences from multiple parts of the state-action space and avoid becoming too narrowly specialized.

d) Robustness and Continuous Improvement: A well-chosen exploration rate not only helps the agent adapt to uncertain or evolving environments but also protects against convergence to local optima. High exploration rates encourage thorough searches of the environment, but they can slow down convergence. Conversely, low exploration rates enable fast exploitation of the current best policy but limit discovery of improvements. Striking a suitable balance or scheduling ϵ to decay over time is often key to making the algorithm robust and capable of continuous improvement, even after many training iterations.

e) Algorithm: Deep Q-Learning (DQN) Outline:

```
1) **Initialize**:
   • Q-network with random weights $\theta$.
   • Target network weights $\theta^- \leftarrow \theta$.
   • Replay buffer $\mathcal{D}$ (empty).
2) **For each episode**:
   a) Observe initial state $s_0$.
   b) **For each time step** $t$:
      • **Action selection:**
        – With probability $\epsilon$, select a random action $a_t$.
        – Otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$.
      • **Execute** action $a_t$, observe reward $r_t$ and next state $s_{t+1}$.
      • **Store transition** $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$.
      • **Sample mini-batch** of transitions from $\mathcal{D}$.
      • **Set target** for each sampled transition $(s_j, a_j, r_j, s_{j+1})$:
          $y_j = \begin{cases} \text{if } s_{j+1} \text{ is terminal:} \\ r_j \\ \text{otherwise:} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^-) \end{cases}$
      • **Update** $Q$-network by minimizing the MSE loss:
          $\mathcal{L}(\theta) = \sum_{j \in \text{mini-batch}} \Big(Q(s_j, a_j; \theta) - y_j\Big)^2.$
      • **Periodically update** the target network:
          $\theta^- \leftarrow \theta.$
```

III. Environment construction

A. Limit Order Book

At the high-frequency level, price movements are largely the result of interactions between supply and demand. Thus, the bid-ask difference can only take on integer tick values and are located at a distance more or less close to $p_{ref} = \frac{p_{bid}+p_{ask}}{2}$. At any given moment, new players can be added to a price, which will then be modeled as a queue, whose order is defined by the order of arrival. The graph above is a theoretical representation of asset queue sizes. We can see that queue sizes get bigger and bigger, and theoretically should increase from 0 to $+\infty$, as a buyer or seller has a greater interest in getting as far away from p_{ref} as possible, to increase profits. This is not observed in practice, however, as standing in a queue is not free, and there is therefore a limit at which the expectation of profit becomes lower than the cost of entry.

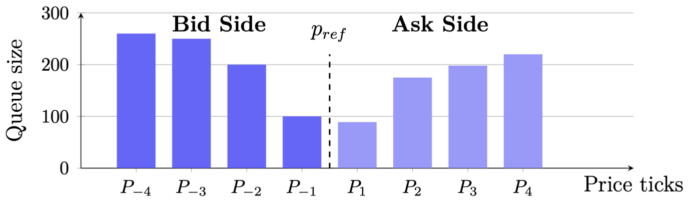

Fig. 1. Theoretical Limit order book

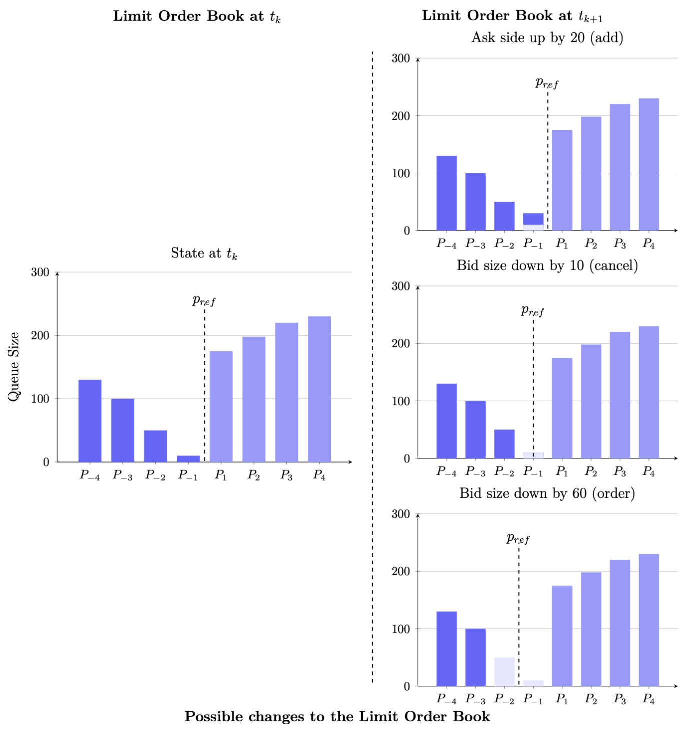

Fig. 2. Different state evolution possibilities

The study of the first limit of the MBO (Bouchaud et al., 2004 and Besson et al., 2016) allows us to understand how the size of the first limit of the bid and ask influence the next price movement. Unfortunately, this prediction is not precise enough to be profitable in the long term, and a more detailed study of the price formation process is therefore required. However, the study of the first limit allows us to extract a first key piece of information from the order book, the imbalance, defined as follows:

$$\text{Imb}_t = \frac{Q_t^{best\ bid} - Q_t^{best\ ask}}{Q_t^{best\ bid} + Q_t^{best\ ask}}$$

.

B. Construction of our environement

The construction of our environment is mainly based on the Queue Reactive model created by M. Rosenbaum and C-A.

Lehalle.

1) Fixed-price QR model: Let p_{ref} be the fixed reference price. We'll model the evolution of the queues around this price. We model the order book by a vector:

$$Q(t) = (Q_{-3}(t), Q_{-2}(t), Q_{-1}(t), Q_1(t), Q_2(t), Q_3(t))$$

which evolves over time according to a Markov process. The element $Q_{\pm i}(t)$ corresponds to the availability of supply or demand at $p_{ref} \pm i$ tick at time t.

To model the MB's next move, we draw $6K$ independent Poisson random variablesof respective parameters $\lambda_i^U(\text{Imb}_t)$ with $U \in \{Add, Cancel, Trade\}$. The next action will then be chosen as an AES (average event size) of action U_{min} at tail $Q_{i_{min}}$ completing the minimum on these Poisson variables:

$$(i_{min}, U_{min}) = \underset{(i,U)\in\{-K,K\times\{A,C,T\}}{\text{argmin}} \mathcal{P}(\lambda_i^U(\text{Imb}_t))$$

The action will be performed at date $t + \mathcal{P}(\lambda_{i_{min}}^{U_{min}}(\text{Imb}_t))$. To ensure that more is added than consumed, we force the add intensities to be lower than the summed cancels and orders.

This model encapulates the basis of the financial market movements at high frequency has it makes different response times considering the current state of the market. For istance, if the ask is greater than the bid, people will tend to go on the bid side as they can leverage their inferiority with better prices.

2) QR model with variable price: We still need to model price movements p_{ref}. To do this, we consider that price p_{ref} can only move under two conditions:

- exhaustion of a limit
- adding a new limit

In all likelihood, the p_{ref} price should move in the direction of the market, i.e. upwards when a limit is exhausted at the ask level, for example. However, this behavior is not observed on the market, and a mean-reversion phenomenon predominates. To capture this evolution, each time the price p_{ref} changes, we draw a binomial distribution with parameter θ, which will be worth 1 if the price follows the market movement and 0 if it goes in the opposite direction. Initial observations by C-A.

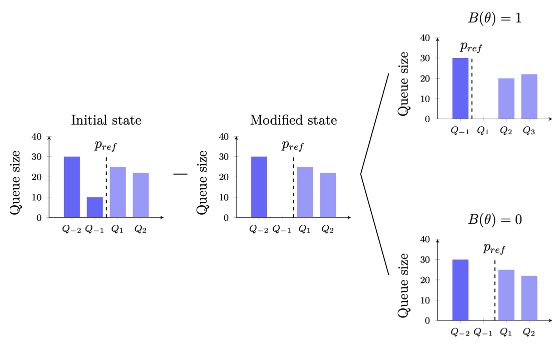

Fig. 3. Price evolution

Lehalle and M. Rosembaum, showed that this model failed to capture actual market volatility. Indeed, even assuming a QR totally driven by the LOB, i.e. $\theta = 0$, the volatility did not reach the value observed in practice. To overcome this problem, they added another parameter $\theta_{reinit} \approx 0.13$, which trigger a price jump with probability θ_{reinit} and which creates a new LOB at the new price. However this modeling is not possible in our case as we need to have a continuity for our agent to be able to interact with the environment.

3) Order size and Exogenous information modeling: In order to recreate the usual volatility seen in HF markets, we need to tweak the QR model. We first consider the possibility not only to make an action of size 1 (1 add, 1 cancel, 1 market order) , but of different sizes taken uniformly randomly between $[1, \text{Size}_{\text{max}}]$, with Size_{max} changing for each action.

To recreate exogenous information, which is extremely important in finance as mentioned by Bouchaud, we had a random event simulation:

- At each step an exogenous event occurs with probability θ_{event}
- If no event occurs we continue with the model developed before
- If an event occurs, we draw its average length (in number of event) from a predefined vector $\nu = [10, 100, 1000]$ for instance. After this draw, we pick the length of our event with a Poisson variable:

$$\text{length}_{\text{event}} \sim \mathcal{P}(\nu[\mathcal{U}(\text{length of } \nu)])$$

- the intensity of each event (add, cancel, market order) instantaneously becomes $\frac{\lambda}{\text{length}_{\text{event}}}$, simulating the higher flow of order during these events.
- The intensity progressively returns to its normal rate with its intensity being $\frac{\lambda}{\text{Nb of remaining events}}$.

These changes makes the market a lot more realistic and will force our agent to incorporate exogenous information in its decision making.

However this model is rather complex and we will not try to fit it with real market data in this paper as it is not the first objective. Other models known are the Bouchaud diffusion model or any tick by tick modeling of the price that incorporates bid-ask demand. Recent studies highlighted the possibility of using Hawkes processes for a better modeling.

IV. Results and Discussion

A. Environment

At first we see the price only has discrete values and moves rather quickly. Comparing it to real data, we see it follows a similar dynamic with a comparable volatility. Moreover it is capable of recreating exogenous information as seen with the 'drop' in Fig. 4. We choose the intensity functions accordingly as the one in *Detection of Exogenous Price Moves: Localizing the Queue-Reactive Model*, C-A. Lehalle, E. Laferté et al.. [3]

With this model we will thus be able to train our agent.

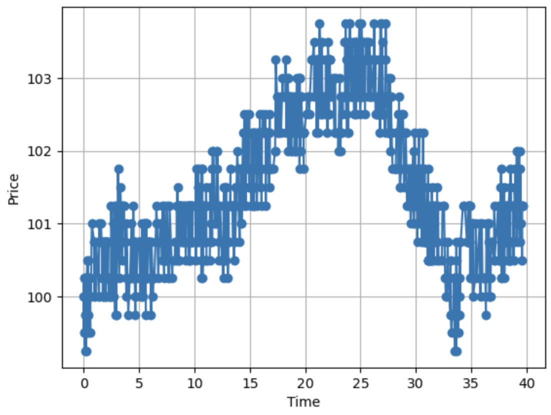

Fig. 4. Price evolution Simulation (without the agent operating)

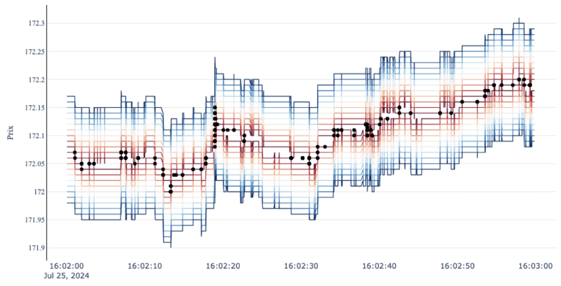

Fig. 5. GOOGL price evolution with bid-ask on 2024-07-25 between 16 : 02 : 00 and 16 : 03 : 00. Black dots correspond to trades, plots correspond to limits from -9 to +9

B. *First results of the Deep Q-Learning*

We then train our agent according to the deep Q-learning algorithm. The reward is chosen as the current gain. The agent start with 1000\$. At first we let our agent interact with the environment between every event, but we rapidly observed it had too huge of an impact on the market, making it insufficient. This directly link with the results of K. Webster in *The Handboook of Price Impact Modeling*[1], showing how over interacting could be destructive and highlighting the need of price impact studying when dealing with HF strategies.

We thus decided to limit its action for now, the agent being able to interact every 50 events. The results are positive, as shown in Fig. 6. We see that the average gain tends to increase after 10000 learning steps, beating the random strategy by a huge margin ($\approx 100\%$). The reward sometimes drops to 0 as the agent has the possibility to do nothing. However, we see it tends to make this decision less and less as the learning progresses.

V. Conclusions

High frequency trading has completely revolutionized financial markets and are essential in portfolio management. We give an entire environment and agent to learn high frequency trading using deep Q-learning. We then show the agent can beat random decision making and seem to improve other time despite the highly stochastic nature of our environment. Future work could include latency and transaction costs as they are important factors in HFT.

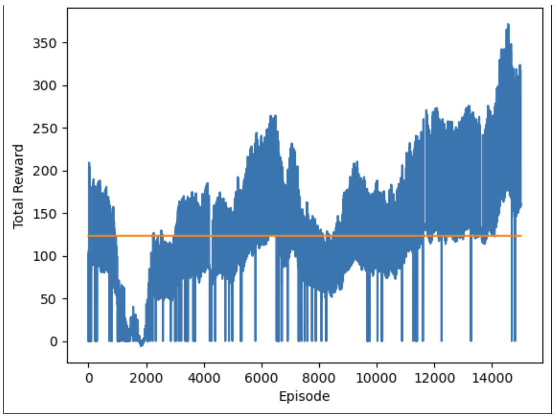

Fig. 6. Agent result compared to a random strategy

References

[1] The Handbook of Price Impact Modeling, K. Webster
[2] Simulating and Analyzing Order Book Data: The Queue-Reactive Model, M. Rosenbaum et al.
[3] Detection of Exogenous Price Moves: Localizing the Queue-Reactive Model, C-A. Lehalle, E. Laferté et al.
[4] Optimal Liquidity-Based Trading Tactics, C-A. Lehalle,
[5] Deep Reinforcement Learning for Active High Frequency Trading, A. Briola)
[6] Unpublished paper, C.A Lehalle, E.Laferté, J. Aida