

An Introduction to Reinforcement Learning in Finance

ZHAW School of Engineering

Fernando de Meer Pardo

September 17, 2022

Introduction to Reinforcement Learning (RL)

RL in Finance

Our project: A Modular Framework for RL Optimal Execution

Conclusions

Iterative Decision Making

- ▶ Reinforcement Learning is a Machine Learning framework in which the objective is to learn rules for **iterative decision** making under **uncertainty** so that a specific numerical quantity, called a reward, is optimized.

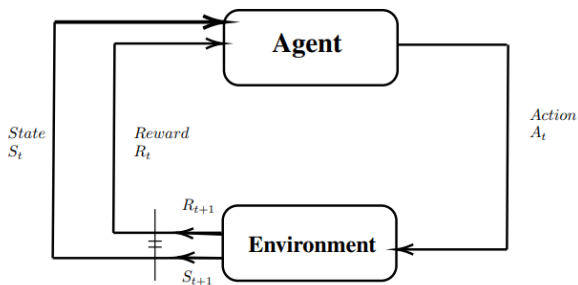
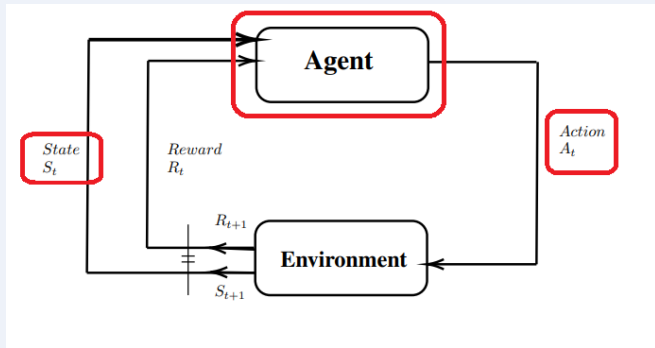


Figure: Basic structure of a RL setup.

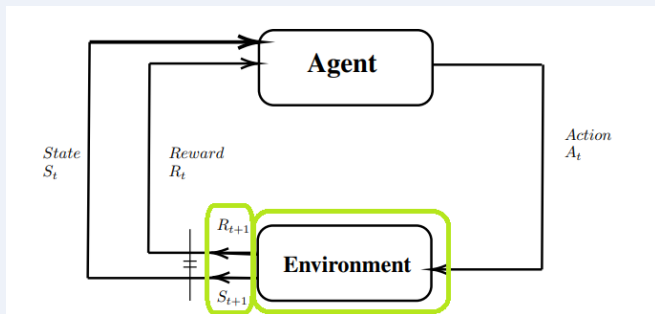
Iterative Decision Making

- ▶ The algorithm tasked with obtaining rules is called the *Agent*; it receives as input the state of the world S_t via another algorithm, called the *Environment*.



Iterative Decision Making

- ▶ The Agent maps states to actions via a special function called the *Policy* π . Given an action A_t taken by the Agent, the Environment produces an updated value of the reward R_{t+1} as well as a new state of the world S_{t+1} .



Iterative Decision Making

- ▶ The goal of an Agent is to find a policy π that maximizes the expected discounted cumulative reward:

$$\arg \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T-1} \gamma^t R_t (S_t, A_t) \right] \quad (1)$$

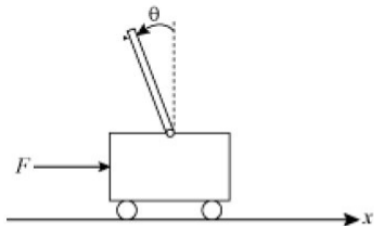
with the $\gamma \in [0, 1]$ being the discount factor, a parameter that reflects the time value of rewards.

There exist multiple algorithms to optimize this objective: Dynamic Programming, Q-values/Value-based methods, policy gradient methods etc.

See: **Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.**

Inverted Pendulum on a cart

- ▶ The inverted pendulum is a classical control problem:
Animation Link
- ▶ The goal is to keep the pendulum upright, the reward is the mean height of the end of the pendulum.
- ▶ Every 1/2s the agent is able to move the cart horizontally. The agent receives as information the angle of the pendulum θ and the position of the cart x



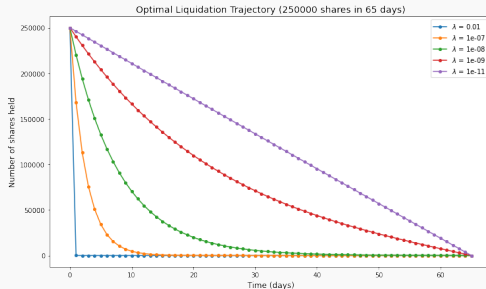
Applications?

- ▶ So this is all very interesting but what can we do with it in Finance? Is there any financial setting that can be tackled via **iterative decision making** under **uncertainty**?



Optimal Execution

- ▶ An **execution** in an electronic financial market context is the process in which a number of units of a publicly traded financial asset are bought or sold through said market.
- ▶ Optimal Execution algorithms aim to carry out this process by dividing the transaction into child orders in such a way that a **series of metrics are optimized**.



Classical Algorithms

A small number of well-known algorithms dominate the commercial offering of execution services. Examples include:

- ▶ Algorithms that attempt to directly capture the **VWAP (Volume Weighted Average Price)** benchmark such as Percentage-Of-Volume.
- ▶ Algorithms that aim to participate in a given % of market volume.
- ▶ **Time Weighted Average Price (TWAP)** algorithms that submit child orders of the same volume at a constant rate.
- ▶ Algorithms that interact with the **bid/ask spread**.

All of these algorithms incorporate a relatively limited amount of market information in their decision-making process and decide order submissions based on relatively straightforward easily explainable rules.

The Deep RL Paradigm

- ▶ Conversely, Deep RL Algorithms can incorporate **arbitrary market data** in their decision-making. They are able to obtain, by **learning from experience** on historical market data, adaptable execution policies that are less susceptible to front-running.
- ▶ The main disadvantages of DRL include the **black-box nature** of the obtained execution strategies and the difficulty that financial simulations, that constitute the training environment for policies, involve.

RL in Finance – Challenges

- ▶ One crucial characteristic of RL applied in market applications is that we do not have access to the true environment and thus have to decide on how to **simulate its behaviour**.
- ▶ In other areas of the RL literature where the Environment can be easily interacted with (for example in physical simulations, games etc.) the usual focus of research is on the Agents and Optimization Algorithms used. This has also been the case with Optimal Execution works.
- ▶ Performance under one simulation setup does not necessarily generalize to other setups and ranking simulation setups by their quality is challenging (or outright impossible). It is thus necessary to be able to implement multiple simulation setups in a serialized way.

Optimal Execution as a RL problem

In order to implement an Optimal Execution RL Environment we need a series of functionalities:

- ▶ Data fetching and pre-processing
- ▶ Construction of observations
- ▶ Action processing
- ▶ Child order execution
- ▶ Simulation of benchmarks
- ▶ Reward calculations

Data fetching and pre-processing

Datasets of High-Frequency (HF) market data can come in many different formats, with different levels of granularity, that allow for different simulation setups. They can be broadly categorized (although there are exceptions in specific markets) into what is commonly referred to as "Levels"

- ▶ **Level 1 data:** Consists of records of traded prices at different timestamps. It may include the very first levels of the LOB, the best bid/ask and their respective volumes as well.

Data fetching and pre-processing

- ▶ **Level 2 data:** Extends Level 1 data by including deeper levels of the LOB, that is, the price levels greater/lower than the lowest ask/highest bid respectively and their volumes. This is usually referred to as "snapshots" of the LOB. They may have a fixed size or change at each timestamp. Often price levels very far away from the best bid/ask are not recorded whereas in a real market this can be a possibility.
- ▶ **Level 3 data:** Consists of the sequence of limit order placements, deletions and executions. It is the most granular out of the three types, since the other two can be reconstructed from it, and can lead to large datasets because of this.

We need to be able to retrieve data iteratively in order to produce observations for the agent.

Action Processing/ Order Executions

- ▶ A RL agent needs to be able to interact with its environment, the market under an Optimal Execution setting, during the training process. In order to implement this feedback mechanism between the market and the agent's actions, translated as order submissions, we need to be able to simulate order executions. We need an implementation able of placing orders into the market, simulating their execution and keeping logs of the trades carried out over the entire trading horizon.

Simulation of benchmarks / Reward calculations

- ▶ Rewards in RL Optimal Execution are often calculated as the difference in performance between the RL Agent and a benchmark algorithm (usually the TWAP). Because of this, we need to be able to record and simulate arbitrary execution algorithms.
- ▶ Each Execution Algorithm divides a transaction into child orders across the execution interval according to some internal logic. The order submission schedule and each order's volume may be decided a priori (like a TWAP algorithm), may be determined via the algorithm's internal state and/or market conditions during the execution (like a %-of-volume algorithm) or may incorporate a combination of both.

RL in Finance – Challenges

- ▶ We need to incorporate representation of each Execution Algorithm consisting of its actions indexed by their timestamps, the variables involved in each action (volume, type of order etc.), as well as variables involved in the particular logic of the algorithm and an internal state containing the execution parameters such as the trading direction, the total volume to execute or the volume left to execute.

Proposed RL Environment Setup

- ▶ We introduce a **modular framework** that allows for the separate implementation of all the different functionalities and internal processes that an Optimal Execution RL Environment needs to incorporate. The core modules we introduce are the *Data Feed*, the *Broker* and the *Execution Algo*

Class Name	Functionalities	Required Methods	Dependency
Data Feed	Data retrieval and pre-processing	reset()	Broker attribute
Execution Algo	Execution Logic	reset()	Broker attribute
Broker	Order Execution Simulation	reset() & simulate_order()	Environment attribute
gym Environment	RL logic	reset() & step()	None

Table 1: Proposed modules of the Optimal Execution RL framework.

Figure: Proposed Modular Framework.

Proposed RL Environment Setup

- ▶ We present an example implementation of a particular Optimal Execution setup through our modular framework. We share the code in a public Github Repository:

https://github.com/FernandoDeMeer/RL_Optimal_Execution

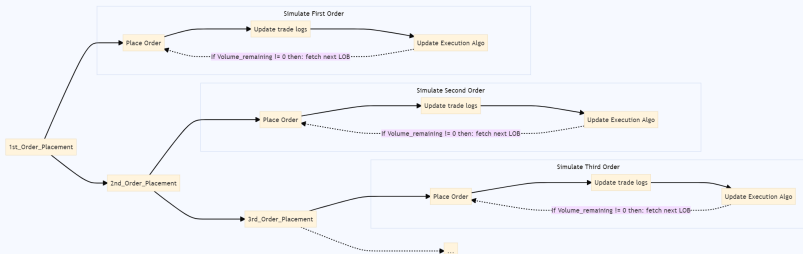


Figure: Illustration of the Order Placements execution simulations.

Algorithm 1 Simulation of a Limit Buy Order Execution

Input: Limit Order O_l , Data Feed D , Execution Algo A , current timestamp t , market tick_size
Output: Execution of order O_l recorded in `Broker.trade_logs`

```
1: while  $O_l$ .volume > 0 &  $t < A$ .next_event.time do:
2:    $O_l$ .price =  $D$ .current_LOB.get_best_bid() - tick_size
3:   Update  $D \Rightarrow D$ .next_LOB()
4:   Update  $t \Rightarrow t = D$ .current_LOB.time
5:   trade_log = place_order( $O_l$ ,  $D$ .current_LOB), append to Broker.trade_logs
6:   if trade_log.message == trade then
7:     Update  $O_l$ .volume  $\Rightarrow O_l$ .volume - = trade_log.volume
8:     Update  $A$ .remaining_volume  $\Rightarrow A$ .remaining_volume - = trade_log.volume
9:     if  $O_l$ .volume = 0 then
10:      break
11:   if  $t \geq A$ .next_event.time &  $A$ .next_event == limit_order then
12:      $A$ .next_event.volume + =  $O_l$ .volume
13:   else
14:     if  $t \geq A$ .next_event.time then
15:       Submit a market order of  $A$ .remaining_volume
```

Algorithm 2 Simulation of a Market Buy Order Execution

Input: Market Order O_m , Data Feed D , Execution Algo A , current timestamp t
Output: Execution of order O_m recorded in `Broker.trade_logs`

```
1: while  $O_m$ .volume > 0 do
2:   Update  $D \Rightarrow D$ .next_LOB()
3:   Update  $t \Rightarrow t = D$ .current_LOB.time
4:   trade_log = place_order( $O_m$ ,  $D$ .current_LOB), append to Broker.trade_logs
5:   Update  $O_m$ .volume  $\Rightarrow O_m$ .volume - = trade_log.volume
6:   Update  $A$ .remaining_volume  $\Rightarrow A$ .remaining_volume - = trade_log.volume
```

Figure: Order Execution Simulation Algorithms.

Proposed Implementation

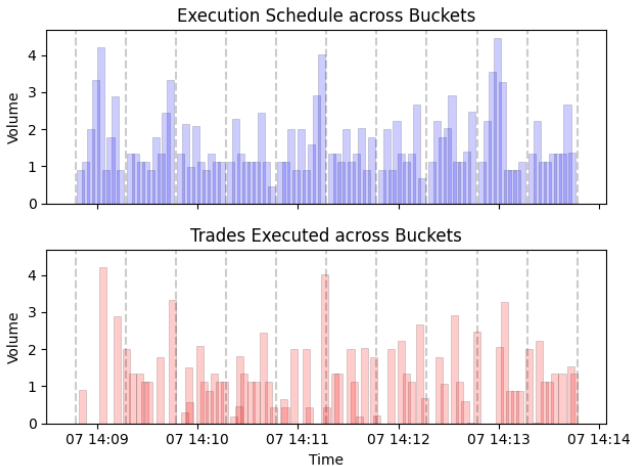


Figure: Illustration of an order submission schedule vs executed trades following the previous Algorithms.

Conclusions

- ▶ Our work can be of interest to both practitioners and academics since it provides a basis that can vertebrate and serialize the implementation of all the aspects of RL Optimal Execution, allowing for a more uniform way to **implement, benchmark and monitor** different simulation setups.
- ▶ Our paper is currently under review for publication in the Open Springer Journal of Financial Innovation and can be accessed on arxiv: <https://arxiv.org/abs/2208.06244>

Thanks for attending!

Please feel free to ask questions.