Executive Summary of the Thesis

Exploiting FX Trading Patterns at Multiple Time-Scales with Hierarchical Reinforcement Learning

Laurea Magistrale in Mathematical Engineering - Ingegneria Matematica

Author: Luca Zerman

Advisor: Prof. Marcello Restelli

Co-advisor: Pierre Liotet

Academic year: 2021-2022

1. Introduction

The Foreign Exchange Market (FX) is a decentralized market for the trading of national currencies made up of a network of banks, institutions and individual traders who operate from all around the world. The FX market is the largest financial market in the world with a daily volume of \$7.5 trillion, ensuring FX and especially EUR/USD traders several advantages that range from high liquidity to low bid-ask spreads. Technology became increasingly important in FX trading in the 1990s with the rise of electronic trading platforms and sophisticated computer systems that allowed traders to access real-time market data and to develop algorithmic trading strategies. Given that the trading process can be modeled as a Markov Decision Process and that the dynamics of the FX market are unknown, Reinforcement Learning (RL) becomes an excellent candidate to design autonomous trading agents. RL has been used in trading since the early 2000s, predominantly in algorithmic trading for options, futures and portfolio management.

In FX, the choice of the trading frequency is an important issue. Actually, the optimal frequency might depend on the market conditions and this is the basis of our work. Indeed, in our thesis we aim to apply Hierarchical Reinforcement Learning (HRL), conceived to learn policies at different acting scales, to FX trading. Our work begins with the implementation of a batch RL algorithm from the literature called Persistent Fitted Q-Iteration (PFQI) [2], which allows to tune the control frequency. Afterwards, PFQI is revisited in what we call Adapted Persistent Fitted Q-Iteration, that represents the fundamental element of the novel algorithm we propose: Hierarchical Persistent Fitted Q-Iteration (HPFQI). HPFQI allows to consider different frequencies simultaneously, training an agent who is able to understand, at each time-step, which frequency is the best, therefore exploiting signals at different scales while keeping a high control on its actions. We first test HPFQI on the Mountain Car, an environment commonly used as a benchmark in RL, and then, after having performed some exploration on our data, we backtest the algorithm on the EUR/USD market. In particular, we consider the highly-liquid EUR/USD currency pair and a relatively small trading size to assume the possibility of quickly going long or short without any market impact and slippage issues. Our results confirm the critical importance of the possibility to constantly change frequency, but also highlight where some improvements could be made.

2. Reinforcement Learning

Reinforcement Learning (RL) is an automatic learning technique that focuses on solving problems where an agent interacts with an environment, learning to make decisions by performing actions and observing the resulting rewards.

2.1. Markov Decision Process

The above scheme is formally representable through a Markov Decision Process (MDP). A discrete MDP is defined as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is the space of the possible states of the environment while $\mathcal{A}$ is the space that contains the actions that the agent can perform. The transition probability function $P(s'|s,a)$ gives the probability of reaching a new state s' from the state-action pair (s,a) and it must be satisfied that the future is not influenced by past history if the present is known, *i.e.* the environment response at step $t+1$ depends only on state and action at time t (*Markov Property*). The reward function manages the rewards the agent collects for that transition and $\gamma \in [0,1)$ is the discount factor, which determines how much importance is given to future rewards. The agent acts through the policy $\pi(\cdot|s)$, which associates a distribution over the action space $\mathcal{A}$ to each state s. Therefore, the agent's goal is to find the policy that maximizes the expected discounted sum of future rewards defined as:

$$J_\pi := \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k R_{k+1} \right],$$

where T is the final trajectory time-step. A useful quantity linked to the expected return is the action-value function $Q^\pi(s,a)$, which denotes the expected return starting from state s, taking action a and then following policy π:

$$Q^\pi(s,a) := \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right].$$

Starting from $Q^\pi(s,a)$, we can define the optimal value function

$$Q^*(s,a) := \max_\pi Q^\pi(s,a), \quad \forall s \in \mathcal{S}\ \forall a \in \mathcal{A}.$$

The policy that selects the action that maximizes $Q^*(s,a)$ is π^*. $Q^*(s,a)$ satisfies the Bellman equation:

$$Q^*(s,a) = \mathbb{E} \left[R_{t+1} + \gamma \max_{a' \in \mathcal{A}(s')} Q^*(S_{t+1}, a') | s, a \right]$$

This equation is useful because it allows us to create algorithms that find the optimal Q-function.

2.2. Fitted Q-Iteration

FQI is a model-free, off-policy and offline algorithm designed to learn an approximation of the optimal action-value function without knowing the transition probability. FQI trains a regressor to fit the Q-function from a dataset $\mathcal{D}$ composed a tuples of state, action, reward and next state, with the aim of generalising over the outcomes not contained in $\mathcal{D}$. In brief:

$$\mathcal{D} = \{(s_t^{(i)}, a_t^{(i)}, r_{t+1}^{(i)}, s_{t+1}^{(i)})\}_{i=1}^{|\mathcal{D}|}.$$

Being FQI offline and off-policy, $\mathcal{D}$ can be collected in a previous phase using any policy. Knowing Q_{N-1} from the previous iteration, Q_N is trained on:

$$Q_N(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a \in \mathcal{A}(s)} Q_{N-1}(s_{t+1}, a),$$

where $(s_t, a_t, r_{t+1}, s_{t+1})$ is a tuple in $\mathcal{D}$. Specifically, at each iteration of the algorithm, the horizon considered increases by one step. A higher number of iterations, even if it guarantees that the agent optimizes for a longer horizon in the future, introduces an overestimation bias: indeed, each FQI iteration implies taking the maximum of an approximated quantity, therefore the Q-function tends to increase artificially with the iterations. For this reason, the number of iterations must be chosen carefully.

2.3. Persistent Fitted Q-Iteration

In RL, continuous-time problems are usually transposed into a discrete-time framework by introducing a time discretization based on a certain control frequency. Clearly, since it is not known which frequency is best a priori, there is a risk that it will be chosen incorrectly. In order to avoid so, the concept of *persistence* is introduced [2], with the aim of finding the optimal control frequency in the trade-off created by the desire of having a high control and a low sample complexity. Indeed, when increasing the control frequency, the advantage of individual actions becomes infinitesimal; as a consequence, the sample complexity increases. Instead, low frequencies allow the environment to evolve longer, making the effect of individual actions more easily detectable. More practically, persistence consists in the repetition of an action for a fixed number of decision steps. If we consider a discrete-time MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, persistence can be seen as an environmental parameter k that transforms $\mathcal{M}$ into $\mathcal{M}_k = \langle \mathcal{S}, \mathcal{A}, P_k, R_k, \gamma^k \rangle$ in which, whenever the agent takes an action, the resulting transition lasts for k steps, with all the one-step rewards collected (with discount) in the new distribution R_k. Applied to FQI gives rise to Persistent Fitted Q-Iteration (PFQI) algorithm.

2.4. Regressors

We consider two regressors in this work: *Extra-Trees* and *XGBoost*.
Extra-Trees essentially builds an ensemble of unpruned regression trees. Its two main differences with other tree-based methods are that it splits nodes by choosing cut-points fully at random and that it uses the whole learning sample to grow the trees. The parameters to be tuned have different effects: the number of attributes randomly selected at each node determines the strength of the attribute selection process; the minimum sample size for splitting a node and the minimum sample size for a node to be a leaf node determine the strength of averaging output noise; the number of trees determines the strength of the variance reduction of the ensemble model aggregation.
XGBoost is a decision-tree-based ensemble method based on gradient boosting. Boosting is an ensemble technique that consists in adding new models sequentially to correct the errors made by the old ones until no further improvements can be made. Gradient boosting is a boosting approach where new models are created in order to predict the residuals of prior models and then added together to make the final prediction; the loss generated is minimized through a gradient descent algorithm. XGBoost differs because it supports stochastic and regularized gradient boosting forms. The most important parameters to be tuned are `min_child_weight`, which controls the minimum number of samples required to be present in each leaf of a tree, and `max_depth`, which controls the maximum depth of each tree.

3. Related Works

3.1. Deep RL for Finance

Deep reinforcement learning (Deep RL) is a promising field of RL for trading due to its capacity to handle complex and high-dimensional data. An example is *DeepScalper* [4], a deep RL framework for intraday trading that, to efficiently incorporate both micro-level and macro-level market information, proposes an encoder-decoder architecture to learn robust market embedding. To capture the overall price trend, a novel hindsight reward function is designed with a long-term profit regularizer to provide the agent with the long-term horizon.

3.2. Hierarchical RL

The Hierarchical RL was designed with the aim of autonomously decomposing long-horizon decision-making tasks into simpler *subtasks*. In the general structure, there is a *higher-level policy* which learns to perform the task by choosing optimal subtasks which in turn may themselves be RL or just *primitive actions*. The hierarchy of policies obtained in this way collectively determines the behavior of the agent. HRL is formalized on the basis of the theory of *Semi-Markov Decision Process* (SMDP) which, unlike MDP, it also involves the concept of time for which an action is executed after it has been chosen. More formally, we can define two important components of the hierarchical framework: the *subtask space* Ω_H and the *hierarchical policy* π_H. The former identifies the super-set of all the subtasks used in a hierarchy, while the latter represents the complete state-to-subtask-to-action mapping from the lowest level policy, *i.e.* the policy that selects primitive actions. Thus, the goal of HRL is to find the *optimal hierarchical policy* π_H^* given a task. HRL has so far churned out few finance-related works, but one is of considerable interest. In [1] the structure is made by a behaviour policy that determines the action to be played given the current state and a skip policy that determines how long to commit to this behaviour. This approach is very similar to ours but actually there is a big difference: by introducing a skip-policy, there is no explicit comparison between the different persistences that can be chosen.

4. Hierarchical Persistent Fitted Q-Iteration

Considering the concept of persistence certainly helps in trading, but we feel that it is not enough: in fact, very often the market abruptly alternates behaviour, making it necessary for the agent to be aware of several frequencies at once. Taking our cue from [5], we decided to create a structure consisting of several hierarchical steps, each of which provides an estimate of $Q(s,a)$ that is increasingly refined with the help of the previous steps.
To achieve so we first define an auxiliary algorithm: Adapted Persistent Fitted Q-Iteration (APFQI). This algorithm differs from PFQI in the definition of the target Q_N. In particular, the term that in PFQI was $\max_{a\in\mathcal{A}} Q^{(K)}_{N-1}\left(s^{(i)}_{t+K}, a\right)$ now becomes:

$$\max\left\{\max_{a\in\mathcal{A}} Q^{(K)}_{N-1}\left(s^{(i)}_{t+K}, a\right), \max_{\substack{a\in\mathcal{A}\\ Q\in\mathcal{Q}}} Q\left(s^{(i)}_{t+K}, a\right)\right\}$$

where $\mathcal{Q}$ represents a set of pre-trained action-value functions. At this point, the derivation of HPFQI (Algorithm 1) is as follows: given a set of persistences $\mathcal{K}=\{k_1,k_2,\dots,k_p\}$ s.t. $k_1 > k_2 > \dots > k_p$, a set of iterations $\mathcal{N}=\{N_1,N_2,\dots,N_p\}$ and a dataset $\mathcal{D}=\{(s^{(i)}_t, a^{(i)}, r^{(i)}_{t+1}, s^{(i)}_{t+1},\dots, r^{(i)}_{t+k_1}, s^{(i)}_{t+k_1})\}^{|\mathcal{D}|}_{i=1}$, HPFQI starts training APFQI at persistence k_1 with N_1 iterations and $\mathcal{Q}=\emptyset$, obtaining $Q^{(k_1)}_{N_1}$ and adding it to $\mathcal{Q}$. At the second step, it trains APFQI with k_2, N_2 and $\mathcal{Q}=\{Q^{(k_1)}_{N_1}\}$, obtaining $Q^{(k_2)}_{N_2}$ and adding it to $\mathcal{Q}$ and so on. Eventually, the policy π_H is retrieved by $Q^{(k_p)}_{N_p}$. The decreasing order of the persistences is chosen in order to both learn the trend easily and eventually have a policy with a high control power.

Algorithm 1 HPFQI

```
1: Input:
   $\mathcal{K}=\{k_1,k_2,\dots,k_p\}$ $k_1 > k_2 > \dots > k_p$
   $\mathcal{N}=\{N_1,N_2,\dots,N_p\}$
   $\mathcal{D}=\{(s^{(i)}_t, a^{(i)}, r^{(i)}_{t+1},\dots, r^{(i)}_{t+k_1}, s^{(i)}_{t+k_1})\}^{|\mathcal{D}|}_{i=1}$
   $\mathcal{Q}=\emptyset$
2: Output:
   Greedy hierarchical policy $\pi_H$
3: for $k\in\mathcal{K}$ do
4:    APFQI($k$, $\mathcal{Q}$)
5:    $\mathcal{Q}\leftarrow\mathcal{Q}\cup Q^{(k)}_{N_k}$
6: end for
7: $\pi_H(s)=\operatorname{argmax}_{a\in\mathcal{A}(s)} Q^{k_p}_{N_p}(s,a)\ \ \forall s\in\mathcal{S}$
```

5. HPFQI on Mountain Car

5.1. Problem Formulation

Although our main goal is to use the HPFQI in the FX market, we decided to test it first in the Mountain Car (MtCar) setting, an environment commonly used as a benchmark in RL. The problem involves a car positioned between two hills with the goal of reaching the top of the one on the right. The car is subject to gravity, thus must build up momentum by repeatedly driving up the left hill and then coasting back down towards the right one. MtCar is easily declinable as an MDP. The state is made of car's position and velocity. The possible actions are: accelerate to the left (-1), to the right (1) or do nothing (0). The reward function assumes value 0 if the agent reaches the goal, -1 otherwise. Finally, the transition probability is deterministic.
MtCar has an episodic structure: specifically, each episode begins with the agent being in a random location within its domain at zero velocity and ends either when the maximum number of steps (l_{ep}) is reached or when the agent reaches the goal within this limit; we set $l_{ep}=256$. The dataset collected at persistence k at each episode is represented as follows:

$$\mathcal{D}_{ep}=\left\{\left(s_{kt'}, a_{kt'}, \sum_{i=1}^{k}\gamma^{i-1} r_{kt'+i}, s_{k(t'+1)}\right)\right\}_{t'=0}^{g/k-1}$$

where g stands for the minimum between l_{ep} and the time-step, if any, in which the goal is reached and the original time-step is nothing more than $t=kt'$. In other words, in each episode, g/k tuples are sampled where in each one the agent repeats the same action for k steps. If the goal is reached in the middle of a persistence step, *i.e.* in a $t\in(kt', k(t'+1))$, the reward of the last tuple sums up to $t-kt'$. The initial training dataset is formed by the union of a certain

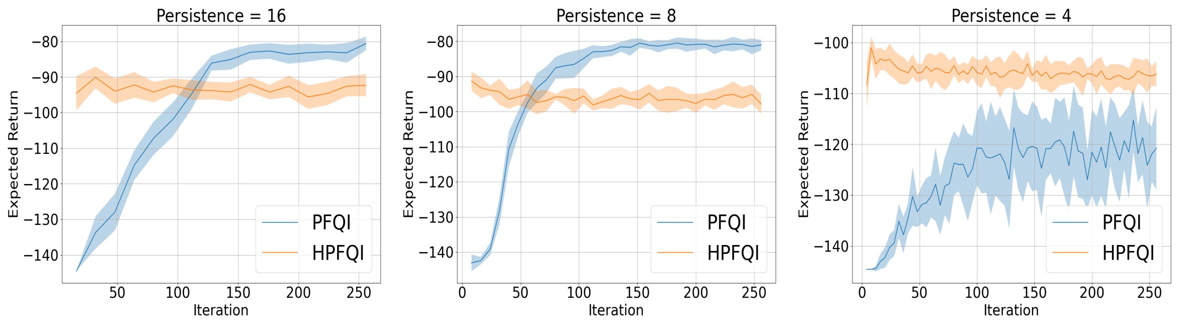

Figure 1: HPFQI vs PFQI in MtCar. The x-axis represents the product between the persistence and the iteration.

number (n_{ep}) of $\mathcal{D}_{ep}$, each one gathered by an agent following a random policy.

5.2. Results

We trained both HPFQI and PFQI in order to compare their results. We chose as persistences $\mathcal{K} = \{32, 16, 8, 4\}$ and as respective number of iterations $\mathcal{N} = \{8, 16, 32, 64\}$ so that each combination has the same level of approximation. To ensure that enough trajectories reach the goal, with $k \in \{32, 16\}$ we considered $n_{ep} = 1000$, whereas with $k \in \{8, 4\}$ we considered $n_{ep} = 5000$; we made this distinction because lower persistences are less likely to reach the goal with random policies. We used Extra-Trees to fit the Q-function. For each $k \in \mathcal{K}$, we retrieved the regressor's best parameters validating PFQI over 20 episodes and used them for training and testing both PFQI and HPFQI.
Ten HPFQI models were trained with $k \in \mathcal{K}$, $n \in \mathcal{N}$ and with an initial dataset collected at persistence 32, all of them tested on 20 episodes. The results are not particularly encouraging, as the expected return decreases as persistence goes by. Specifically, at the immediate jump between two persistences there is a noticeable performance loss, while as the number of iterations of a single persistence increases the expected return remains about the same/slightly worsens, except of course for the first one. To better understand these results, for each $k \in \mathcal{K}$ ten PFQI models were trained on their respective datasets and tested on 20 episodes. From a comparison of the results (Figure 1), we notice two main behaviours: in the *initial iterations*, HPFQI reaches a higher expected return than PFQI, which states that HPFQI is able to exploit the knowledge of the higher persistences. At *the final iterations*, instead, almost always PFQI performs better than HPFQI. Notably, with $k = 4$ HPFQI at each iteration is better than PFQI for both return and variance, symptom that the hierarchy makes the algorithm more robust to noisy situations.
To understand more deeply these results, we performed further analyses that brought interesting considerations (Figure 2): firstly, at the transition between two persistences, many new states are seen on which the previous Q-function struggles to adapt. The main cause is probably that most of the new states are inevitably far from the goal, thus lowering the proportion of states close to it. Secondly, as persistence decreases, the values assumed by the Q-function on average both increase and accumulate in a central area: this may happen due to the overestimation of the Q-function along the iterations. Despite the poor results of HPFQI on MtCar, we believe that the FX environment might be a better evaluation task for HPFQI given that sampling is not necessary as we use historical data and the rewards

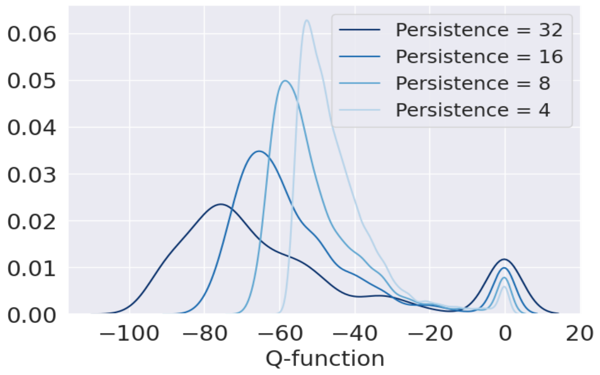

Figure 2: Density representation of the Q-function at the last iteration of each persistence of HPFQI.

are not sparse as in MtCar.

6. HPFQI on FX

6.1. Problem Formulation

The original dataset we were able to retrieve from HistData online platform is made of market observations collected every minute from Monday to Friday. In addition to the *date* and *time*, each observation included the *mid price* p_t, *i.e.* the average between the bid price, the highest price a buyer will pay, and the ask price, the lowest price a seller will accept, and the value of the bid-ask *spread* σ, *i.e.* the difference between the bid price and the ask price. To avoid overnight fees and to parallelize environment sampling, we decided to represent the trading framework as an episodic task, where each episode is composed of daily data between 8:00 CET and 18:00 CET. Since we assume that the agent can only deal with a fixed quantity of asset, the possible actions are: buy (1), sell (-1) or be flat (0). The state is composed of the minute of the day, the day of the week, the spread σ_t and the portfolio position x_t. It also contains the last 60 normalized exchange rate differences, defined as $d_t^k = \frac{p_{t-k+1}-p_{t-k}}{p_{t-k}}$ for $k = 1, 2, \ldots, 60$. Finally, the reward function is $R_{t+1} = A_t(p_{t+1}-p_t)-\frac{1}{2}\sigma_t|A_t-x_t|$: the first term represents the pure profit/loss that the agent incurs taking action A_t, whereas the second one indicates the costs associated to that transaction. The sampling method differs from MtCar's on-policy one. Here, the uncontrollable part of the state comes from historical data, while the controllable part is the portfolio x_t, which corresponds to the last action a_{t-1}. Having three possible actions, there exist only nine combinations of (x_t, a_t). Therefore, one can collect an extensive dataset:

$$\mathcal{D}_{ep} = \left\{ \left(s_t, a_t, \sum_{i=1}^{k} r_{t+i}, s_{t+k} \right) \right\}_{t=0}^{g-k-1},$$

where, in our case, $g = 600$.

6.2. Data Exploration

Before training our model, we wanted to check the stationarity of d_t^k on the years 2018 and 2019. To do so, we computed the autocorrelation of the average of d_t^k, without normalization, over the days. Since the autocorrelation showed significant dangerous repetitive patterns, we decided to check more carefully whether this spurious signal could actually affect our agent or not. To recover the exact size of the spurious signal, we retrieved the square root of the absolute value of the autocovariance K. On the other hand, the agent's training is affected if the reward function is significantly altered by the signal, *i.e.* if $p_{t+1} - p_t \gg \frac{1}{2}\sigma_t$. Fortunately, given that $\sqrt{|K|} < \frac{1}{2}\mathbb{E}[\sigma_t]$, this does not occur. Furthermore, we computed the average volatility of the data over the days: firstly, the evidence of higher volatility between 8:00 and 18:00 CET positively supports our choice for the considered trading period. Secondly, some repetitive volatility spikes are shown, probably caused both by the openings/closings of international markets and by automated traders. Whatever the reason, we believe that HPFQI can be influenced by these only when working at persistence 1, since with higher persistences often they will not be considered in the dataset tuples. In conclusion, we believe that the dataset cannot adversely affect the performance of our algorithm, thus we can proceed with the HPFQI experiments on FX.

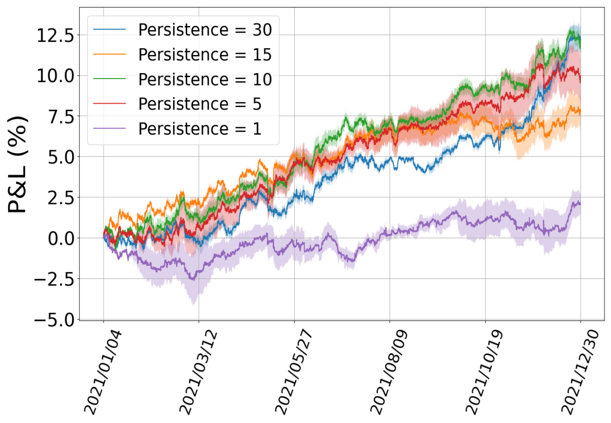

Figure 3: HPFQI($\mathcal{K}_3$) P&L obtained in test after training on 2018/2019.

6.3. Results

The parameters used for training both PFQI and HPFQI were obtained by training PFQI on 2018/2019 and validating on 2020, not considering the months of March and April which, because of the COVID19, generated an anomalous pattern. Specifically, the parameters we varied were `min_child_weight` and `max_depth`.

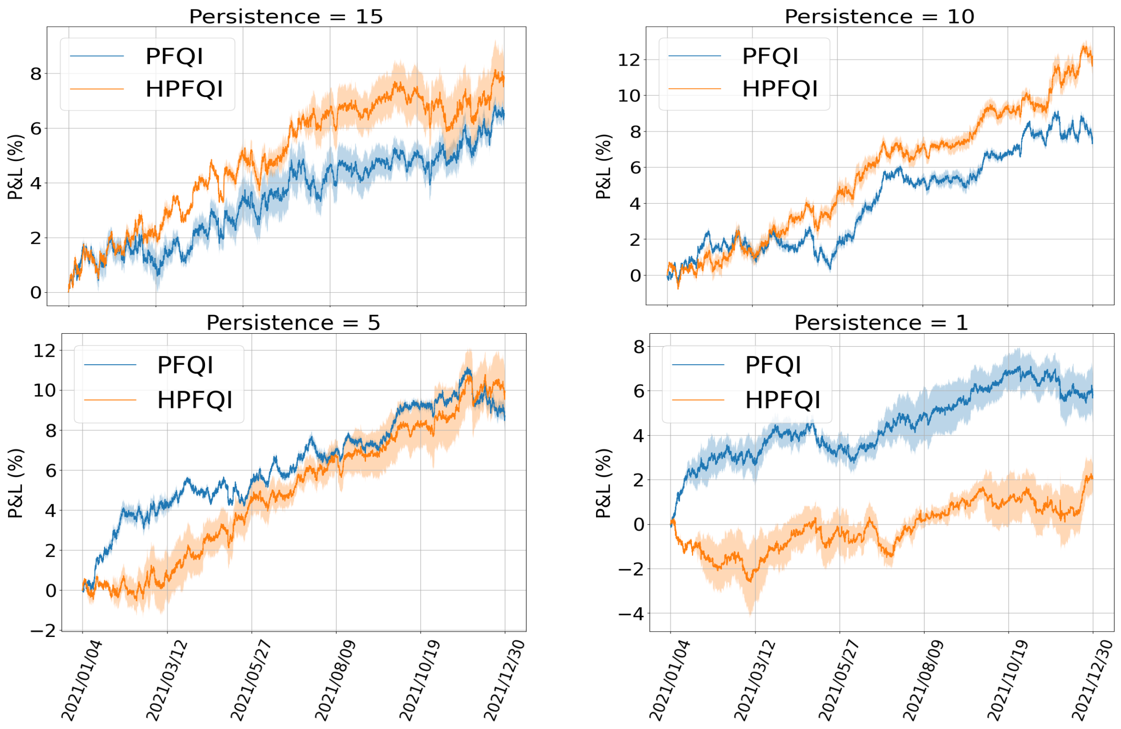

Figure 4: HPFQI($\mathcal{K}_3$) vs PFQI P&L obtained in test after training on 2018/2019.

The combinations of persistences considered are $\mathcal{K}_1 = \{10, 5, 1\}$, $\mathcal{K}_2 = \{15, 10, 5, 1\}$ and $\mathcal{K}_3 = \{30, 15, 10, 5, 1\}$, whereas we set the same number of iterations ($N = 8$) for each of them. To avoid validated parameters being affected by a change in dataset length, both HPFQI (Figure 3) and PFQI were trained on 2018/2019 and tested on 2021; in all the cases three seeds were used. In all three cases, we found similar behaviors: at the first one/two persistences of the hierarchical chain, the profit obtained outperformed PFQI's at the respective persistence (Figure 4). Indeed, HPFQI profit almost always shows an upward trend, whereas PFQI's more often flattens out. However, after taking several hierarchical steps, HPFQI loses all the advantages gained from transferring Q-function and sometimes even performs worse than PFQI. Looking at the actions performed by the agents, several interesting aspects can be seen (Figure 5). Firstly, HPFQI agent tends to buy more. Secondly, HPFQI policy is always more organized than PFQI's: indeed, the former displays repetitive patterns both hourly-wise and daily-wise, while the latter is more chaotic. This difference is probably the main cause of the results, both positive and negative, obtained by HPFQI: indeed, at the earliest persistences HPFQI's policy is certainly structured but still leaves room for improvisation in case of anomalous trends. At the latest ones, instead, the hierarchy imposes an over-structured policy. This last aspect is confirmed by the feature importance analysis: at the latest persistences, the HPFQI agent exploits almost only date and time variables without looking at the actual exchange rate itself.

7. Conclusions

In this thesis, we have implemented a novel algorithm called Hierarchical Persistent Fitted-Q Iteration (HPFQI) to train an artificial agent to autonomously trade in the FX market. Three are the main advantages with respect to [3]: the introduction of a new HRL algorithm (HPFQI) that allows the agent to encapsulate both lower and higher-frequency information, the former facilitating the understanding of the effect of an action and the latter giving more control to the agent; the detailed quantitative inspection of the FX data used for training and testing our algorithm; the implementation of the XGBoost algorithm as the regressor used to approximate the Q-function in PFQI. HPFQI was designed to exploit a hierarchy of auxiliary algorithms, APFQI, sequentially trained at differ-

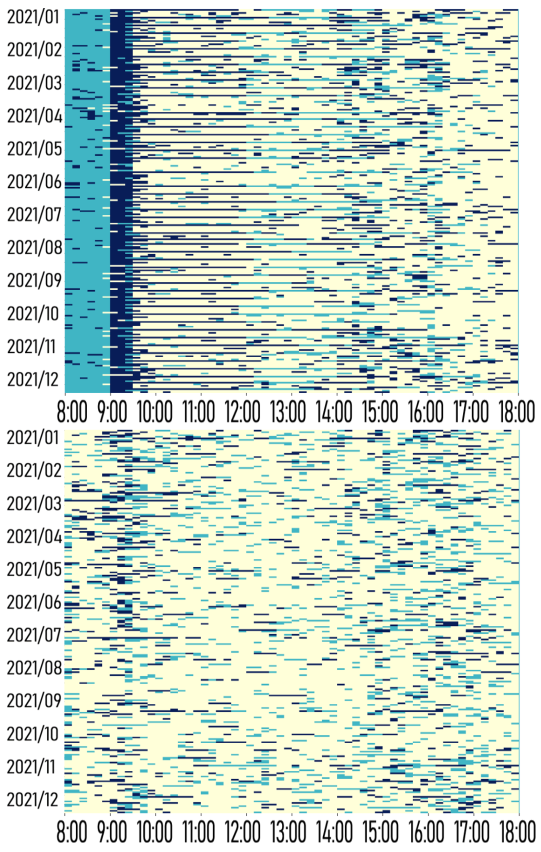

Figure 5: Heatmaps of the actions taken by HPFQI($\mathcal{K}_3$) (above) and PFQI (below) with $k = 10$. Each row represents a trading day and each column a minute. Blue = buy; light-blue = flat; yellow = sell. It is evident that HPFQI's agent has a policy more organized and buys more than PFQI.

ent frequencies. First, we tested HPFQI on the Mountain Car environment, where we had mixed results: it is evident that the passage of information between the various frequencies has taken place but the overall performance deteriorates as the number of iterations performed by the algorithm increases. Therefore we carried out more in-depth analyses, which showed that the performance drops are probably caused both by the difficult adaptation of the Q-function to the many new states introduced and by the overestimation of the Q-function after many iterations of PFQI are performed.
However negative the empirical results have been, they were very informative. Moreover, we believed that the FX environment might be a better evaluation task for HPFQI given that sampling is not necessary as we use historical data and the rewards are not sparse as in Mountain Car. Before getting the results, we decided to do some analyses on the specific EUR/USD dataset. In particular, we noticed unexpected autocovariance patterns in the dataset that were shown however to not give lucrative information to the agent.
Training and testing HPFQI on different FX frameworks we have noticed how, in general, it is able to exploit information from previous persistences to better organise its policy. In several cases, this factor resulted in a sometimes remarkable improvement in the P&L with respect to PFQI. However, when the hierarchical chain becomes too long, the policy often becomes overstructured, generating profits lower than PFQI ones.
Regarding potential future research directions, one way to reduce the overestimation problem could be to use the Double Q-Learning method within the hierarchical algorithm. Another approach might be to include information from higher persistences' actions in the state.

References

[1] André Biedenkapp et al. Temporl: Learning when to act. *CoRR*, 2021.

[2] Metelli et al. Control frequency adaptation via action persistence in batch reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[3] Riva et al. Learning fx trading strategies with fqi and persistent actions. In *Proceedings of the Second ACM International Conference on AI in Finance*. Association for Computing Machinery, 2022.

[4] Sun et al. Deepscalper: A risk-aware reinforcement learning framework to capture fleeting intraday trading opportunities. In *Proceedings of the 31st ACM International Conference on Information &amp; Knowledge Management*. Association for Computing Machinery, 2022.

[5] Tiancheng Zhao and Mohammad Gowayyed. Algorithms for batch hierarchical reinforcement learning. *CoRR*, 2016.