Reinforcement learning for financial markets
Eric Benhamou
Université Paris Dauphine
eric.benhamou@dauphine.eu

Abstract

The main hypothesis underlying most financial markets models is market efficiency. Market prices should fully reflect all available information and it should be impossible to systematically capture profits in financial markets.

In practice, human agents’ decisions are affected by some behavioral biases. These biases affect the way agents perceive and mis-assess of probabilities that make the game not as efficient as thought.

In this ongoing research, we investigate the creation of an intelligent trading agent that adapts to the complexity of financial markets and performs profitable trading. We cast the general problem into a stochastic control problem and explore two traditional Reinforcement Learning algorithms to solve it (Q-Learning and SARSA) as in [Bertoluzzo-Corazza (2015)]. We extend their results by taking a full matrix representation. We present results that shows that RL algs outperform a simple buy and hold strategy.

1. Primer on RL

Investing in financial markets is not easy. As prices roll up and down, choosing a long or short position is a complex task, stressful and risky. It can be very emotionally up and down, choosing a long or short position is a complex task, stressful and risky. It can be very emotionally

Reinforcement learning (RL) is an area of machine learning inspired by behaviourist psychology, that helps reformulating the decision problem as a dynamic programming problem where an agent needs to take actions so as to maximize some notion of cumulative reward (for instance see [Bertoluzzo (2017)]. The general setting of Reinforcement learning is summarized by figure 1 below.

![Image of a figure showing classical agent-environment interaction](image)

**Figure 1:** Classical Agent-Environment Interaction

At each discrete time step, \( t = 0, 1, 2, \ldots, N \), the agent:
- receives a reward \( r_t \in \mathbb{R} \) as a response to its previous actions.
- perceives the current state \( s_t \in \mathcal{S} \) of the environment.
- selects an action \( a_t \in \mathcal{A}(s_t) \) where \( \mathcal{A}(s_t) \) is the set of available actions in state \( s_t \).

The agent’s objective is to maximize the sum of its total discounted rewards denoted by \( J_\pi \), given by:

\[
J_\pi = \sum_{k=0}^{N} \gamma^k r_{k+1} \quad \text{where} \quad \gamma \in [0, 1] \quad \text{is the discount rate}.
\]

This problem is classically reformulated in terms of a Q matrix as follows. At each step \( t \), the agent, starting from the state \( s_t \), has to calculate the value of the action-value function \( Q^\pi(s, a) \) for each possible action \( a \) based on the policy \( \pi \):

\[
Q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{N} \gamma^k r_{k+1} \mid s_0 = s, a_0 = a \right].
\]

A fundamental property of the action-value function is the fulfillment of the following special recursive relationship, known as Bellman equation for \( Q^\pi(s, a) \):

\[
Q^\pi(s, a) = r(s, a) + \gamma \max_{a'} Q^\pi(s', a'),
\]

This Bellman equation has a unique solution that can be computed numerically with various algorithms.

2. Q-Learning & SARSA algorithms

Two traditional algorithms to solve the Bellman equations are Q-Learning and SARSA. They compute recursively the Q matrix thanks to a stochastic gradient approach:

\[
\Delta Q_t(s, a) = \alpha [y_t - Q_t(s_t, a_t)]
\]

where \( \alpha \in [0, 1] \) is the learning rate and the step is in the sense of a gradient that differs in the two methods. In Q-learning, we take the “optimistic” view that all future action selections from every state should be optimal; thus we pick the action \( a^* \) that maximizes \( Q_t(s_t, a^*) \). This leads to an optimal state given by the next step reward and the optimal decision:

\[
\text{Step} = s_t + \epsilon \text{rand} \quad \text{and} \quad \text{NewStep} = Q_t(s_t, a^*) + \gamma \text{MaxStep}\]

In SARSA, the agent learns \( Q_t(s_t, a_t) \) on the basis of the performed action, hence the step is simpler to compute and given by:

\[
\Delta Q_t(s, a) = \alpha [y_t - Q_t(s_t, a_t)]
\]

In the trading environment, one has to define the world and the reward. We follow [Bertoluzzo-Corazza (2015)] and take the last \( N = 5 \) logarithmic returns. The reward is the traditional Sharpe ratio over the last 20 days (corresponding to a full month), we take a greedy step \( \gamma = 0.95 \) and transaction brokerage cost equal to \( 0.20\% \) of the stock value. We take as a benchmark the full matrix approach parametrized by \( v \) as follows

\[
Q_t(s_t, a_t) = r_t + \gamma \sum_{k=1}^{N} v_k (s_{k+1}, a_{k+1}) + \beta_k X_{t+20}(m_t)
\]

where \( v_k \) is a logistic function. Our improvement is to realize that we can also take what we call a ‘full matrix’ approach and store all \( v \) matrix coefficients given by \( Q_t(s_t, a_t) \).

As \( v_k \) takes \( 5 = 243 \) possible values, and \( v \) takes 3 (short, neutral, long), we have a 243 rows columns matrix. This is largely manageable. We show that this provides better results as there is no more approximation of the Q matrix.

3. Results and Discussion

To compare with [Bertoluzzo-Corazza (2015)], we investi- gate the value of telecom italia for a period from January 1986 to December 2017. Data are from yahoo finance. The results are summarized in table 1. We can see that the full matrix approach is superior. For reference, a buy and hold strategy leads to a performance of \(-6.9\%\). This shows that reinforcement learning improves substantially decision.

**Table 1: Results for the various models**

<table>
<thead>
<tr>
<th>Method</th>
<th>Annual Return</th>
<th>Trades/year</th>
<th>Drawdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sarsa</td>
<td>2.23 %</td>
<td>2.19</td>
<td>-50 %</td>
</tr>
<tr>
<td>Full Sarsa</td>
<td>8.53 %</td>
<td>0.61</td>
<td>-47 %</td>
</tr>
<tr>
<td>Q Learning</td>
<td>2.31 %</td>
<td>2.25</td>
<td>-25 %</td>
</tr>
<tr>
<td>Full Q Learning</td>
<td>9.76 %</td>
<td>10.7</td>
<td>-35 %</td>
</tr>
</tbody>
</table>

On the graphics below, we can notice various facts. First, full matrix representation improves results. Second, it leads to generate more trades. This emphasizes that in this method, it is critical to keep transaction cost at the mini- mum. Third, this comes at the price of more drawdown. If performance is measured by recovery factor (the per- formance divided by the drawdown), the enhancement re- duces.

**Figure 2:** Linear Q-Learning MC: For the various Monte Carlo simulations, we plot the equity line path for the action Telecom Italia. We can see that we have some path ending with negative performance but the majority of the paths are with positive performance

**Figure 3:** Linear Q-Learning Results: The graphics above provides the prices of Telecom Italia, the optimal action and the resulting equity. The method is profitable as the equity line regularly increases. The trading system obtained tracks only occasionally compared to the full matrix ap- proach.

**Figure 4:** Full Matrix Q-Learning MC: Same graph as for the Q linear approximation method but using full matrix one. Note the full matrix provides higher Monte Carlo path, indi- cating a more efficient method

**Figure 5:** Full Matrix Q-Learning Results: Compared to linearized QLearning, the full matrix method gives higher results (average annual return of 9.76 percent compared to 2.31 percent for the linearized Q). It also generates more trades as it is more responsive to states environment

Python source code and data for the experiment can be found on github [Benhamou (2018)]

4. Conclusion

Our improvement to compute the full matrix leads to better performance both for Q and Sarasa learning. This improve- ment comes at the price of more frequent trades and hence larger drawdowns that makes the enhancement questionable in practice. In further research, we would like to add additional states to capture situations where it is not opti- mal to trade because of past performance.

References


