CentraleSupélec

ESSEC BUSINESS SCHOOL

Agent Based Learning for Trading

Yahya JANBOUBI, Ghali LARAQUI HOUSSAINI
DSBA M2
CentraleSupélec, Université Paris-Saclay
Reinforcement Learning
Professor Stergios Christodoulidis

Abstract:

In this study, we explore the potential of reinforcement learning (RL) for algorithmic trading within the volatile cryptocurrency market, focusing specifically on Bitcoin. Utilizing a dataset of over 3 million rows spanning from January 2015 to April 2021, we design and test five different RL agent-models across a range of time frequencies to investigate their ability to uncover profitable trading patterns.

Through a comprehensive literature review, we ground our methodologies in established RL techniques, including Q-Learning, Deep Q-Networks (DQN) while also integrating advanced neural network architectures like double dueling architectures. Despite our sophisticated approach, initial results suggest that models based solely on price signals struggle to consistently outperform the market, highlighting the complexities of financial markets and the need for further innovation in the application of RL and deep learning techniques to algorithmic trading.

Keywords : Algorithmic Trading, Reinforcement Learning, Deep Q-Networks, Dueling Networks, Bitcoin.

Section I - Introduction

Financial services, and particularly market trading activities, have always relied upon and generated a vast amount of very diverse data. The interdependencies observed in the movements of prices of different asset classes, and the impact of macroeconomic data releases, geo-political events, social networks such as Twitter, and even weather conditions, on price discovery patterns and cross-assets correlations dynamics make this field burgeoning with information. The edge that market participants have lies in how they leverage the informative content of the data available - in general - to the whole, and as computational methods grow more complex, it is increasingly hard to develop an all-winning system that beats the market across time and market *zeitgeist* (i.e. market *mood*).

Naturally, thus, the field of market finance has been nudged towards that of Artificial Intelligence. Biological cognitive limitations of the human brain, due to limited attention span, difficulty in performing complex tasks instantly, the need for sleeping, and our innate and inherent emotional nature, have made the concept of 'intelligent machines' an ideal to replace human market participants, driving more efficiency and gains for those who implement them *correctly*. It is not surprising, thus, that the execution of market orders by humans has been replaced electronically by machines, which, with time, optimized the trading process. And while Ken Griffin, Citadel's founder and chief executive officer, claims that Generative AI won't be as good of a trader as experienced market veterans, history shows that the craziest ideas can end up being reality sooner than expected.

Given this context, this project, thus, is an attempt to develop a low to medium frequency algorithmic trading bot which is structured and trained under a reinforcement learning paradigm. The dataset collected is fairly simple. We gathered prices, in U.S. dollar, of the main cryptocurrency in the market, Bitcoin, and the data frame is shaped under the classic *ohvc* format - 'open', 'high', 'low', 'close'. We also included volume indicators, 'from' and 'to', indicating incoming or outgoing flows, respectively. Note that the data is collected at the 1-minute interval, and encompass the period from January 2015 to April 2021. Overall, and after basic processing, we have more than 3 million rows for 6 features. We aim to leverage this high granularity to see whether our models can uncover hidden patterns to best play the market.

This paper is structured as follows: in the next section (Section II) we will quickly review some of the most prominent literature onto which we base the architecturing of our agent-models, before delving deeper into our methodology and programming structure in Section III. The five agents we designed will be described in detail in Section IV, after which we will present under Section V our main results under our trading simulations. Finally, in Section VI we will discuss the limitations of our modeling and approaches that we implemented in this project, and mention some interesting paths to further this work and improve the performance of our agents.

Section II - Literature Review

Reinforcement Learning in the domain of algorithmic trading represents a cutting-edge approach where computational agents learn optimal strategies to execute trades with the goal of maximizing financial returns, in our case profits, over time. This application of RL encompasses the adaptation to a dynamic and uncertain environment, leveraging the capability of agents to make sequential decisions based on the current market state and the expected future rewards. The integration of RL into algorithmic trading has spawned a rich body of research focused on various model architectures and their efficacy in navigating the complexities of financial markets.

One of the foundational methods in RL, Q-Learning, has seen adaptation into the financial trading realm, enabling agents to learn the value of actions in a given state without requiring a model of the environment. The evolution of Q-Learning into Deep Q-Networks (DQN) by Mnih et al. in their seminal 2015 paper *"Human-level control through deep reinforcement learning"* introduced the use of deep learning to handle high-dimensional state spaces. This advancement allowed for the processing of extensive market data, enabling more nuanced trading strategies that consider a broader range of market conditions. Moreover, Policy Gradient Methods and their subsequent evolution into Actor-Critic models have provided an alternative approach by optimizing the policy function directly. This suite of methods, exemplified in the work of Sutton and Barto in "Reinforcement Learning: An Introduction," focuses on learning a policy that dictates the trading actions to be taken, offering a way to manage the continuous action spaces found in trading environments, such as deciding on the size of trades.

Further refinements in RL methodologies have led to the development of Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO). These advanced policy gradient methods, highlighted in Schulman et al.'s 2017 paper "Proximal Policy Optimization Algorithms," focus on improving the stability and robustness of the learning process, which are critical for adapting to the volatile nature of financial markets. In parallel, and given the sequential and temporal nature of financial data, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks have been integrated into RL frameworks to better capture time-dependent patterns in market data. Indeed, Fischer T. and Krauss C. in their 2018 paper "*Deep learning with Long Term Short-Term Memory Networks for Financial Market Predictions"*

This approach facilitates the decision-making process over sequences of trades, addressing the intrinsic time-series aspect of financial market data.

The application of these RL models in algorithmic trading has covered various areas such as portfolio optimization, where RL algorithms dynamically adjust asset allocations to maximize returns; order execution strategies, where the focus is on minimizing market impact and slippage; and market making, where RL agents learn to profitably balance the bid-ask spread while managing inventory risk. Despite the promising results, RL in algorithmic trading faces several challenges, notably the non-stationary and noisy nature of financial markets that demand a high degree of robustness and adaptability from the models. As well, the dilemma of exploration versus exploitation is particularly pronounced in this domain, as is the need to consider regulatory and ethical implications of automated trading strategies.

Section III - Methodology & Programming Structure

Given our data frame is 3 million rows long, as the prices are collected minute by minute, we believed it was judicious to resample Bitcoin prices (closing prices) to create more environments and see whether time granularity has an effect on our agents' performance. To this end, we created 5 data frame variants, with frequency of 5 minutes, 15 minutes, 1 hour, 6 hours, and 1 day. Our initial assumption was that, by lowering the frequency, we decrease the prevalence of noise, and if there is worthy information in past prices in the sense of predictive power from past price only, then we will see an improvement in performance across all our deep learning models as we lower the time frequency.

With regards to our numerical agents, we designed two basic models and three deep learning models. The first two work as benchmarks to the three others. Indeed, instead of taking a raw scalar number, for instance “percentage return for current trading period”, we believed it more informative to test our deep learning models against basic strategies. The first basic model is essentially a random walk (which is the model that surprisingly performs the best in Foreign Exchange markets), and the second is a momentum following strategy. Essentially, the latter takes an average of the previous cut-out prices and uses it as a threshold against which current price is compared. If price > threshold, use buy-like actions, else, use close-like actions (this is a simplification of the actual code). The other three models’ architectures are discussed in more depth in Section IV.

Agents evolving in environments take actions - else all this would be pointless, right? We thought of designing two action spaces: one simple one, with either BUY, HOLD, SELL, or CLOSE_ALL to be chosen, and one complex one, with either BUY, DOUBLE_BUY, COMBO_BUY, HOLD, SELL, DOUBLE_SELL, CLOSE_ALL. BUY and SELL involve the transaction of one unit only (unit here is 1 bitcoin, regardless of the price). DOUBLE is for doubling the transaction (2 units here). COMBO is 6 transactions at once. Finally, CLOSE_ALL is to close all ledgers.

What is interesting in our approach is that we do not give hard, arbitrary limits to the inventory size of our agents. While the reward mechanism can enforce some risk management behaviours, we believed it was better to let the agents build their positions, and see how their performance evolves from there. We still wanted to include a nudge towards going long, which explains why we have COMBO_BUY and not COMBO_SELL. The function that deals with all the ‘accounting’ behind the agent’s positions and action decision is called “transform”, and updates several ledgers, as seen in the file “UTIL_STRUCTURES.py”.

Additionally, training and testing occurs in a ‘randomized’ fashion. We first define our trading period parameter (default is 1000 timestamps), and randomly select a 1000-rows long subset of our dataframe. We then perform training on either 70% or 80% of that data, and test on the remaining 30% or 20%. This was done for both saving training time, but also for generalizability. There is no point in training our model with half of our dataset (whatever the time frequency), and then test it on the remaining length. Markets are alive, dynamic, and so there is more performance information when we randomize the subset taken, then keep on testing on the same data subset (that is, from 2015 to 2019 train, and 2019 to 2021 test). This allows us to see how our model tackles different market structures.

In parallel, and as mentioned in the literature, it is useful for the agent’s learning process to provide him with a Memory, which enriches the representation of the current state (current state is the current price and the previous 13 prices). The memory saves the agents’ previous positions, actions, states, and rewards. Mechanism of the ReplayMemory class can be found in “UTIL_STRUCTURES.py”.

Furthermore, we also included some logic related to the tradeoff of exploration-exploitation. Essentially, we start with a very high epsilon (.99), allowing solely exploration by our agent, and, if the number of steps done by our agent is higher than say 300, we set epsilon to its end value (.12). We did not formally hypertune the end value, but we did want to keep some level of randomness given how erratic/volatile and unpredictable markets can be.

Finally, the reward attribution is fairly simple (but can be complexified as we designed our code to be *very* modular in its structure). If profit is negative, then reward is -10. If profit is positive, then reward is +10. If the position is flat, reward is -2 (to push the agent to continue playing and not just sit tight when profit gets positive). One can also activate the aggressiveness argument, which pushes the agent to trade even more given a specific position (LONG, SHORT, and FLAT).

Section IV - Modeling & Deep Architectures

When dealing with a high number of potential states and actions, the traditional tabular approach becomes unwieldy and impractical. By merging Q-Learning with deep neural networks (DNNs), DQN leverages the superior approximation capabilities of DNNs to replace the need for a table storing Q-values.

DQN operates through two DNNs to enhance learning stability. The main neural network estimates Q-values for the current state-action pair (Q(s, a; θ)), while the target neural network mirrors the main network's architecture to estimate Q-values for the next state-action pair (Q(s', a'; θ')). Learning primarily occurs in the main network, while the target network remains frozen for several iterations. Subsequently, the weights of the main network are transferred to the target network, improving the accuracy of its estimations.

The updated form of Bellman's equation incorporates Q-functions parameterized by network weights θ and θ'. The loss is defined as the squared difference between the two sides of the Bellman equation. For our model, we employed a convolutional DNN architecture to effectively extract features from raw input data, optimizing performance and robustness in complex environments. We specify that the input is the state and the action (state is a vector of size 14: the actual price and the last 13 prices)

$$Q(s, a; \theta) = r + \gamma \max_{a'} Q(s', a'; \theta')$$

$$L(\theta) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta))^2]$$

The Dueling DQN divides Q-values into two components: the value function V(s) and the advantage function A(s, a). The value function V(s) predicts the reward from state s, while the advantage function A(s, a) assesses the superiority of one action over others. By summing V and A, Q-values are obtained. This modification is beneficial as it allows focusing solely on learning the state-value function in certain scenarios where knowing the exact value of each action is unnecessary.

The algorithm proposes a neural network with its last layer split into two segments. One segment estimates the state value function V(s), while the other predicts the advantage function for each action A(s, a). These components are then combined into a unified output to estimate Q-values.
the equation Q = V + A, given Q, determining the values of V and A becomes "unidentifiable.
To resolve this issue, the paper suggests a slight modification to this process.

$$Q(s, a) = V(s) + (A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a))$$

We adopt two variants of Dueling DQN. First, the Linear Dueling DQN algorithm streamlines computations by employing linear transformations to combine these components, making it particularly suitable for scenarios where simplicity in model architecture is preferred or when computational resources are limited.

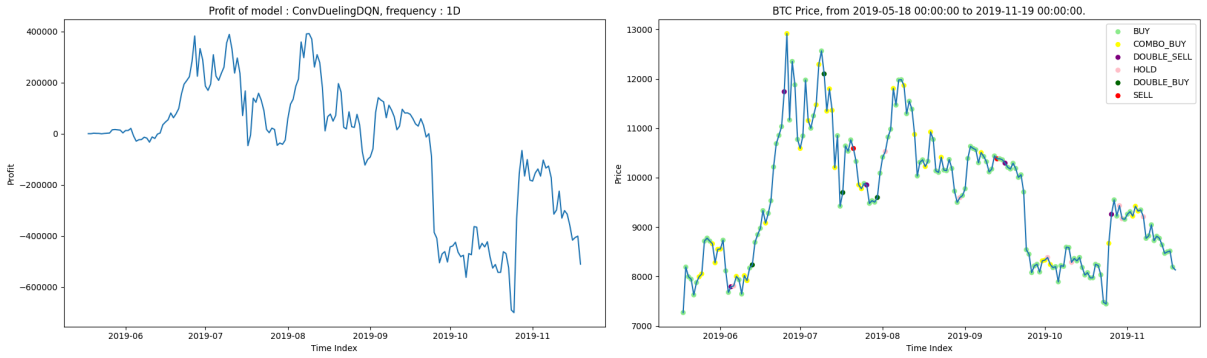
Secondly, the Convolutional Dueling DQN integrates convolutional layers within the dueling architecture, offering a synergistic blend of both methodologies. Leveraging convolutional operations, it efficiently extracts features from raw input data, facilitating effective learning and generalization. By separating the estimation of state values and action advantages, this algorithm strikes a balance between complexity and performance, catering to a wide array of tasks where spatial relationships and hierarchical representations play crucial roles.

Section V - Results & Discussions

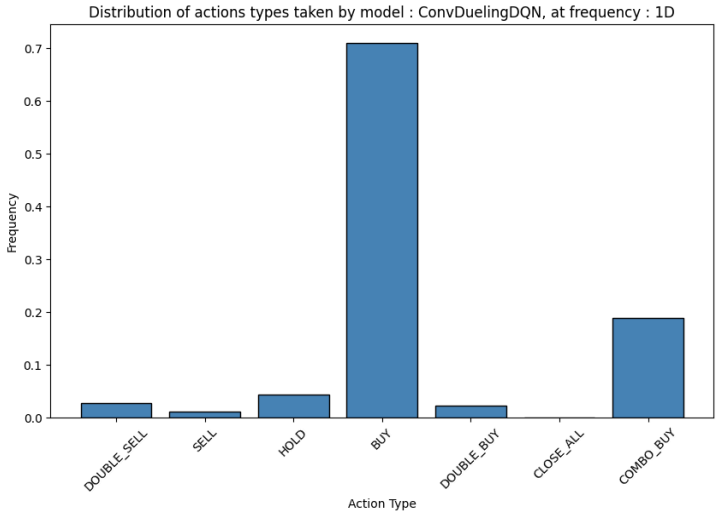
We ran approximately 50 simulations (5 agent-models, 5 environments, 2 actions spaces) and can confidently conclude that none of our models can systematically beat the market. Unfortunately, it seems that solely relying on price signals is not enough to yield consistent returns, even if the asset under question is, on appearance, decorrelated from any macroeconomic and geopolitical news.

In Figure 1 located in the Appendix can be seen a summary of our model's performances varying time frequency for the complex action space. The ranking shows that, despite being the agent that loses the least in all time frequencies, ConvDuelingDQN is actually terrible where all others yield high profits (in the 1D context). To analyze this further, let us look at the graph of the profit for the model, the price action, and the distributions of the actions that it took.

Graph 1 : Profit of Model ConvDuelingDQN, 1D, and BTC Price for corresponding period



Graph 2: *Distribution of the Actions taken by ConvDuelingDQN, 1D, for the same time period.*



It is clear that the model is biased towards the long side. How come is this bias formed? Two preliminary answers we have are : 1) it is linked to the price action of the training set which taught the agent that Buying is the best action; 2) it is reinforced by how epsilon is set at its end-value during testing (.12) rather than higher, which prevents understanding that SELL and DOUBLE_SELL are actually better in this scenario.

This case is actually seen repeatedly across our results, where agents are focusing entirely on a subset of actions, disregarding the rest that would have been better. The reason is that it expects the environment to behave in the same way with the same rules than with the training data frame subset.

But that is what is hard with markets - while history tends to repeat itself, it never does in the same way. Please note that all results for all simulations are located in a folder named “Agents Results”.

Section VI - Limitations & Further Work

While this work is relatively promising and bases itself on state of the art reinforcement algorithms, there are two key limitations that must be addressed to make it more reliable and capable of going ‘in production’ for live implementations.

1) Poor Feature Engineering:

By only considering Closing Price signals (and memory) as our state, we lose on a variety of rich indicators (fundamentals, technicals, behavioral, cross-asset correlations, etc...) that help human market participants in their actions. These indicators are part of the ‘rule of the game’, and if our model cannot learn from them and act upon them, then it misses a big part of the picture.

However, including other features changes the dimensionality of our state, and inputs for our models. In fact, it would require some preprocessing with LSTMs and Attention Mechanisms units to improve the representation of the closing price. This is however hard to implement in practice in a Seq2Seq framework, where the output of the LSTMs-based Encoder will be the input to our Dueling Architectures. Note that in our folder you can find a file with a preliminary attempt to develop a Seq2Seq architecture to achieve just that.

2) Risk & Position Management:

We encountered on several occasions instances where our profits would drop (or rise) significantly in a very short amount of time. That is both because of a strong price movement, and also because of high inventory.

The solution to this would be to implement:

a) “Profit gradient” monitoring: if it rises higher than a certain level, CLOSE_ALL then follow the momentum.
b) Limits to the size of the open positions.
c) Adjusting rewards based on profit behavior and inventory size.

Section VII - Conclusion

In conclusion, results are both promising, and also disappointing. Our strategy to test the performance of our models at various time frames proves to be a good intuition, as our agents tend to be more profitable as the granularity of the data decreases. This makes sense, as there is less noise within our data, and thus price signals hold more ‘true’ information about potential true direction. At high granularity however, it seems that actions taken are either out of sheer randomness, or completely biased towards one specific action type (for e.g. sell), disregarding others. This may be due to how we set up our epsilon-greedy mechanism, as our models would decrease their exploration, a fatal mistake considering how erratic and mercurial markets are.

Overall, we developed a good structure that, with the inclusion of more features of alternative data and technical indicators, can be consistently profitable. To say the least, this was a great introductory project to the field, one we enjoyed a lot. Thank you !

REFERENCES

1. Lucarelli, Giorgio, and Matteo Borrotti. "A Deep Reinforcement Learning Approach for Automated Cryptocurrency Trading." 2019: 247-258.
2. Mnih, Volodymyr, et al. "Playing Atari with Deep Reinforcement Learning." CoRR, abs/1312.5602, 2013.
3. van Hasselt, Hado, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." CoRR, abs/1509.06461, 2015.
4. Wang, Ziyu, et al. "Dueling Network Architectures for Deep Reinforcement Learning." 2015, arXiv:1511.06581.
5. Roach, Jeff, and Nathan Janos. "1D Convolutional Neural Networks for Time Series Modeling." YouTube. https://www.youtube.com/watch?v=nmkqwxmjwzg.
6. Santhosh, S. Thanikam. "Reinforcement Learning Part 4: Dueling Double Deep Q Learning with TensorFlow." Medium, Medium, [no publication date given], https://medium.com/@sthanikamsanthosh1994/reinforcement-learning-part-4-dueling-double-deep-q-learning-with-tensorflow-3f46e65fb644.
7. Reddit User: AdministrativeBank48. "Does It Make Sense to Use RL for Trading?" Reddit, 24 February 2023, https://www.reddit.com/r/reinforcementlearning/comments/10v3o40/does_it_make_sense_to_use_rl_for_trading/.

APPENDIX

Figure 1 - Rankings of Models in terms of Time Frequency