Reinforcement Learning for Stock Trading

Syed Saad Hussain
Department of Mechanical and Industrial Engineering
Ryerson University
Toronto, Canada
saad2.hussain@ryerson.ca

I. Introduction

[1] For my project I developed and solved a toy stock trading MDP model using different reinforcement learning algorithms, and compared the results for the different approaches. My goal was to train an agent to learn optimal trading policies that maximizes trading profits over a fixed time horizon.

Financial markets are complex non-stationary environments with a plethora of factors influencing their behaviour and dynamics over time. In order to train an actual agent to trade profitably in real-time, there are dozens of indicators of market health that may need to be considered. Furthermore, the actual dynamics of a market evolve over time which means that a truly accurate MDP representation should be non-stationary.

However, for the purposes of the project, I simplified the problem immensely. In order to represent the model as a stationary entity, a long time horizon was utilized to take advantage of the notion that blue-chip stocks generally appreciate in value over time. Of course, this means that this toy model is in no way an accurate representation of how one would profitably trade stocks in the real world.

Here is an overview of the different algorithms employed during the experimentation phase:

- First-visit Monte Carlo with tabular representation of action-values
- First-visit Monte Carlo with Linear value function approximation for action-values

Through conducting different experiments the agent was able to match the buy-and-hold strategy, and even outperform it under certain circumstances.

In the following sections, an overview of the MDP model and its inspirations along with detailed analyses of the methods, results, and experiments will be discussed in detail. The flaws of representing this problem using the chosen MDP model will also be discussed, along with potential future considerations.

II. Problem Statement and Environment

I did not use any off-the-shelf environments for my project. I developed and coded the entire MDP model, although it was loosely inspired by the FinRL [1] paper. In the model, the agent starts by owning an arbitrary amount of cash, plus another arbitrary amount of holdings for one particular stock. At the end of each trading day, the agent decides whether it should buy or sell one unit of the stock. At each decision epoch, the portfolio appreciates or depreciates in value. The goal of the agent is to maximize the trading profits when it reaches the end of the time horizon.

Here are the aspects that completely define my MDP:

- State: Price at each decision epoch or time step, p_t. Depending on the experiment, the decision epoch may be the end of each trading day, week, or month.
- Actions: $S = \{buy, sell, hold\}$ For some experiments, the action space was limited to $S = \{buy, sell\}$
- Rewards: The change in portfolio value at each decision epoch, r_t.
- Policies: Describes whether the agent buys, sells, or holds at each decision epoch. It is denoted by π.
- Action-values: $Q_\pi(s, a)$, represented by table or approximated using linear value function approximation depending on experiment.

The biggest limitation of the model is that it only allows the agent to buy or sell one unit of the stock. This is a gross oversimplification of how an investor would trade in the real world, since it is likely that they would prefer to trade multiple units of the same stock at a time.

The total portfolio value (PV) at any given time can be defined as:

$$PV_i = C_i + (n_i \times \mu_{i,p}) \tag{1}$$

where C_i is the value of the cash in the portfolio, n_t is the number of holdings at the current decision epoch, and $\mu_{t,p}$ is the average purchase price of the stock. The value of the stock holdings (v) in the portfolio at a specific time step, t, is formally defined as:

$$v_t = \frac{\sum_{i=0}^{t-1} n_i \times p_i}{\sum_{i=0}^{t-1} n_i} \tag{2}$$

Furthermore, the reward is the change in portfolio value over consecutive decision epochs, and can be defined as:

$$r_i = (C_i - C_{i-1}) + \sum_{t=0}^{t-1} n_i \times (p_t - \frac{\sum_{i=0}^{t-1} n_i \times p_i}{\sum_{i=0}^{t-1} n_i}) \tag{3}$$

To better understand how the different elements of the MDP come together, here is an example. Suppose that the following is true:

- $C_i = \$100$

[1]GitHub Link to Project Files: https://github.com/saad-huss/CP8319-Final-Project

- $p_i = \$50$
- Current action, a_i is $hold$
- Policy, π, up to the current state was $[buy, buy, buy]$
- States, $p = [\$25, \$30, \$40]$

In this hypothetical scenario, PV_i and r_i can be defined as:

$$PV_i = 100 + 3 \times (\frac{25 \times 1 + 30 \times 1 + 40 \times 1}{3}) = \$195 \quad (4)$$

$$r_i = (100-100)+3\times(50-\frac{25 \times 1 + 30 \times 1 + 40 \times 1}{3}) = +\$55 \quad (5)$$

In this case, the action of holding the stock led to an increase in portfolio value by \$55 because the current stock price is greater than the average stock price contained within the portfolio. If the current price dipped below the average price of the holdings in the portfolio, the same action of $hold$ would have resulted in an immediate negative reward.

The goal of the agent is to maximize the portfolio value by the time it reaches the final decision epoch.

III. Experimental Benchmarks

Stock price data at close were obtained through the $yfinance$ library on python. The chosen time horizon for analysis was January 1st, 2010 to December 31st, 2019. A ten-year time horizon was chosen so that the MDP could be approximated as a stationary model. The time period may account for various crucial factors such as bull markets, bear markets, and volatility.

Two stocks from different industries were chosen for analysis to test for significant variances in profitability using the different reinforcement learning algorithms:

1) Apple (AAPL), a large tech company
2) Home Depot (HD), a large industrial company

During the ten-year period, AAPL stock price went from \$7.64 to \$72.88, appreciating 853% in value. HD stock price went from \$28.67 to \$217.31, appreciating 658% in value. Since both companies experienced tremendous growth during the decade, it may be likely that a policy that randomly buys, sells, and holds could still be profitable.

Thus, the benchmark against which the performance of the different methods will be tested is the average return for each stock over thirty random policies. This will provide a lower bound against which model performances will be measured. The upper limit benchmark for comparison will be the traditional buy-and-hold strategy, which is popular among retail investors. Although the buy-and-hold strategy may not theoretically be the most profitable, it provides a reasonable upper bound against which learned trading strategies can be compared. Here is a summary of the benchmarks for the two stocks, assuming that the investor begins on day one with \$10,000 cash and 1000 holdings.

TABLE I
Benchmark Returns

Ticker	Random Policy	Buy-and-Hold
AAPL	\$71,495	\$75,240
HD	\$183,650	\$198,640

IV. Experimental Results

A. *Monte Carlo Methods - Tabular*

The first algorithm I implemented was the first-visit Monte Carlo with tabular action-values. For this MDP, first-visit and every-visit Monte Carlo are identical because no state is ever visited more than once, since the system only progress forward with time.

During the ten-year time horizon, the number of states is 2515 (number of trading days) with 3 potential actions (buy, sell, or hold). Due to hardware restrictions, it is computationally infeasible to traverse through a 2515×3 table over thousands of iterations. Thus, the state space was limited so that the agent only has the freedom to trade once per month. This reduced the action-value table dimension down to a more reasonable 126×3.

The first experiment was to test the influence of different ϵ decay strategies. This plays a huge role in performing ϵ-greedy policy improvement. That is, it has significant impact how much freedom an agent is given to explore, before it exploits its findings from the exploration phase. The two ϵ-decay strategies in the experiment were:

1) Naive exponential decay
2) Naive linear decay

The update rule for naive exponential decay rule was designed as follows:

$$\epsilon \rightarrow \frac{\epsilon}{k} \quad (6)$$

where k is the iterator. The challenge with this decay strategy is that the agent does not have much opportunity to explore since the ϵ value drops significantly during the first few iterations. Below is a chart of how the policy evolves over the number of iterations, using the naive ϵ decay strategy:

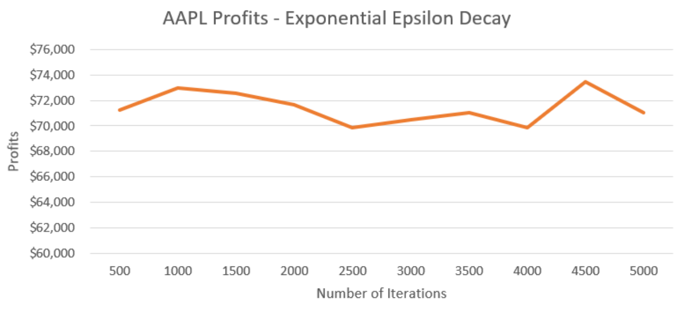

Fig. 1. AAPL Performance - Exponential Epsilon Decay

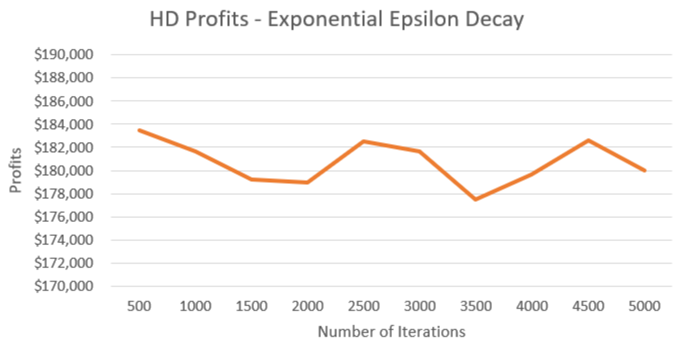

Fig. 2. HD Performance - Exponential Epsilon Decay

As evident from the charts, the policy fails to evolve over the first 5000 iterations, for $\gamma = 0.9$. In fact, the calculated profit after 5000 iterations is no better than the random policy. Thus, it is clear that the agent failed to learn.

The update rule for naive linear decay was designed as follows:

$$\epsilon \rightarrow \epsilon - rate \tag{7}$$

$$rate = \frac{\epsilon_{initial} - \epsilon_{final}}{iterations} \tag{8}$$

where $\epsilon_{initial} = 1$ and $\epsilon_{final} = 0.01$. This linear decay strategy produced much better returns over the same number of iterations, for $\gamma = 0.9$, as seen below:

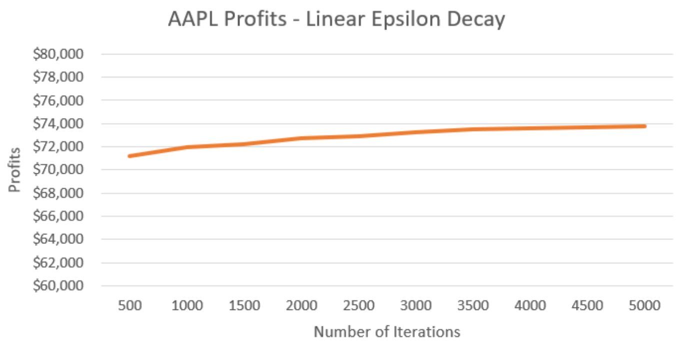

Fig. 3. HD Performance - Linear Epsilon Decay

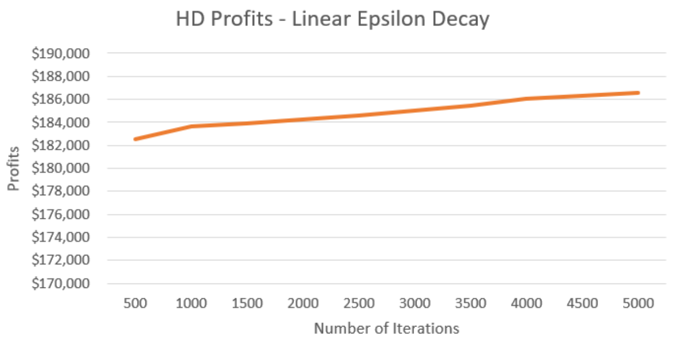

Fig. 4. HD Performance - Linear Epsilon Decay

The next set of experiments were fine tuning the discount rate, γ, to test its influence on the policy. A high discount factor creates an increased emphasis on prioritizing future returns, whereas a low discount factor is biased towards short-term rewards.

For this MDP model, a low discount factor may be represent an aggressive investor who is more eager to cash in on short-term profits at the expense of long-term gains. Conversely, a high discount factor may represent an defensive investor who is more concerned about long-term gains. Below is a summary table with experimental results for different discount factors.

TABLE II
GAMMA VS. RETURNS

Gamma	AAPL Returns	HD Returns
0.1	$75,898	$199,254
0.5	$75,773	$200,157
0.9	$76,029	$200,319

According to the results, the gamma factor does not impact total profits in this toy MDP model. However, the results generally outperform the buy and hold strategy.

The final set of experiments for tabular MC methods was testing the influence of trading fees on the policy and the profits. For this experiment, the reward function, $r_i(3)$ was modified as follows:

$$r_{i,new} = \begin{cases} r_i - fees & a = \{buy, sell\} \\ r_i & a = \{hold\} \end{cases}$$

In reality, commission fees are either fixed per trade (eg. $5) or depend on the number of shares traded (eg. $0.05 per stock purchased or sold). Due to the limitations of the MDP, neither option works since the agent can only buy or sell one stock at a time. It does not make sense to assign a fixed cost of $5 or $0.05.

Instead, the fees were assigned as a percentage of the value of the trade. For example, if a share was purchased for $25, then there would be a cost of $2\%\ of\ \$25 = \$0.50$, which is a tiny amount compared to the overall portfolio value. Since any reasonable percentage would create negligible influence on reward, it was set to an unreasonably high amount for the purposes of analysis. This experiment is unrealistic for real world stock trading, but it does confirm some interesting properties.

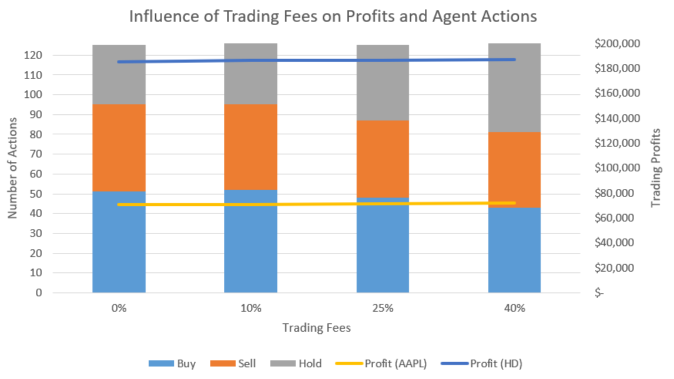

Fig. 5. Model Performance - Trading Fees

There are two interesting insights that can be gleaned from the figure:

1) Impact of trading fees on actions taken
2) Impact of trading fees on profits

Since the agent only has the freedom to trade once per month the total number of actions is bounded by 126, which represents the total number of decision epochs. As the trading fee percentage increases the number of buys and sells shrinks and the number of holds increases slightly. This makes sense because there is no additional cost to holding, whereas there is a cost to buying or selling. The ratio of buys to sells remains relatively flat.

The most interesting finding is that is that the profits actually increase as the trading fees go up. Although this may seem counter-intuitive, there may be an explanation for this. As the trading fees increase, the agent has more incentive to hold a stock. Thus, the total profits tends to approach those achieved by the traditional buy-and-hold strategy.

As mentioned, the tabular MC method limits the number of actions that an agent can take. It would be interesting to see how the agent would behave if it had the freedom to trade every single day during the 10-year time horizon. In the following sub-section, I will discuss the results obtained through representing the action-value function via linear value function approximation.

B. Monte Carlo Methods - Linear VFA

I implemented linear VFA for policy evaluation using the pseudo-code provided in the lecture slides. I wanted to teach the agent to detect historical trends in the price of the stock and use that information to decide whether to buy or sell. For the purposes of this analysis, trading fees were set to \$0.

Without any trading fees, the agent has more freedom to maximize profits by buying and selling as it sees fit. There is no benefit to holding if it does not cost anything to buy or sell a stock. For example, suppose that trading fees were still in place. If the agent only stands to make a profit of \$1 by selling, then that trade is still worth executing. However, if there was a fixed cost of \$5 associated with the trade, it would incur a loss and holding would be a better alternative. Due to these reasons, the number of actions was limited to two, buy and $sell$.

To build the features, recent trends in the movement of the stock price were implemented. More specifically, the prices at the previous three decision epochs were considered. I attempted to design the features in two different ways in order to capture two different investment philosophies:

1) Momentum investing - buying more as the price of the stock rises, and sell as it dips. This method compares the current price to the average price of the previous three decision epochs. We want the agent to recognize that it is a good thing to sell when the current price is less than what the trend suggests, or also a good thing to buy if the current price is higher than the recent trend.
2) Buying the dip - buy more as the price dips, and sell as it rises. This method also compares the current price to the average price of the previous three decision epochs. We want the agent to recognize that it is a good thing to buy when the current price is less than what the trend suggests, or also a good thing to sell if the current price is higher than the recent trend

Here is a mathematical representation of how the features were designed to teach the agent to buy the dip:

$$x(s, buy) = \begin{cases} p_i - \frac{p_i + p_{i-1} + p_{i-2}}{3} & \text{if } p_i > \frac{p_i + p_{i-1} + p_{i-2}}{3} \\ \frac{p_i + p_{i-1} + p_{i-2}}{3} - p_i & \text{if } p_i < \frac{p_i + p_{i-1} + p_{i-2}}{3} \end{cases}$$

$$x(s, sell) = \begin{cases} \frac{p_i + p_{i-1} + p_{i-2}}{3} - p_i & \text{if } p_i < \frac{p_i + p_{i-1} + p_{i-2}}{3} \\ p_i - \frac{p_i + p_{i-1} + p_{i-2}}{3} & \text{if } p_i > \frac{p_i + p_{i-1} + p_{i-2}}{3} \end{cases}$$

In the first expression, an $x > 0$ is assigned if the agent sells upon a price increase. The magnitude of the feature value at that state depends on how rapidly the price has increased during the previous three decision epochs. Similarly, an $x < 0$ is assigned if the agent sells upon a price decrease. The expression for $x(s, sell)$ is the opposite.

For example, suppose the sequence of prices is $[25, 26, 27]$ and the agent sells. In this case, $p_i = 27$ and

$$x(s, sell) = 27 - \frac{25 + 26 + 27}{3} = +1 \tag{9}$$

$$x(s, buy) = \frac{25 + 26 + 27}{3} - 27 = -1 \tag{10}$$

The results were very good. The total profits far exceeded the buy-and-hold strategy by as much as 12%. In fact, the agent almost perfectly learned to buy and sell as per the intention of the feature design. For example, observe the charts below for AAPL and HD, respectively. Note that each chart represents an arbitrary 50-day subset of the 10-year period:

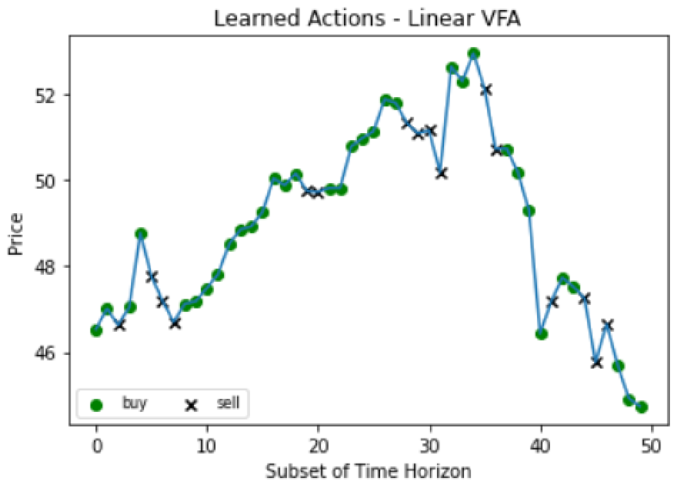

Fig. 6. AAPL Momentum Investing - Linear VFA

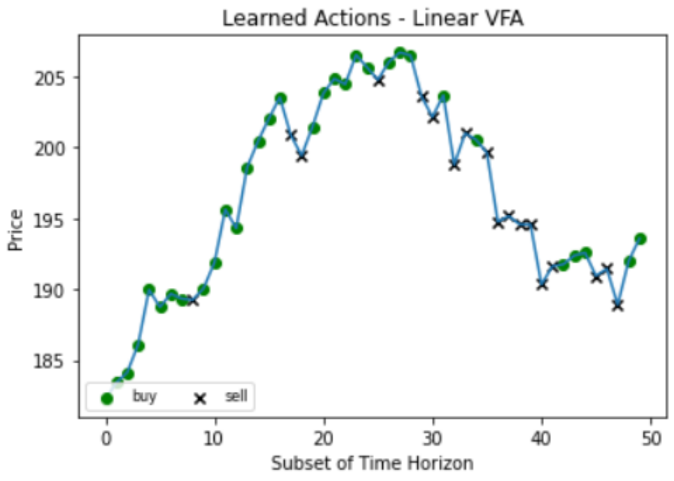

Fig. 7. HD Momentum Investing - Linear VFA

In this case, the agent has successfully learned to act as a momentum investor and buy as the price rises and sell as it falls, purely based on short term stock price trends. However, there are a few small sections where the strategy does not work too well, however, that has to do with the fact that the price is fluctuating wildly around those decision epochs.

The charts below represent an agent that has been trained to buy the dip:

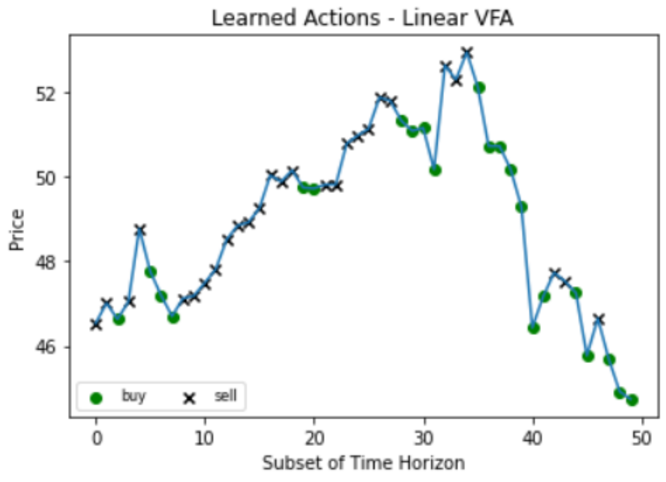

Fig. 8. AAPL Buying the Dip - Linear VFA

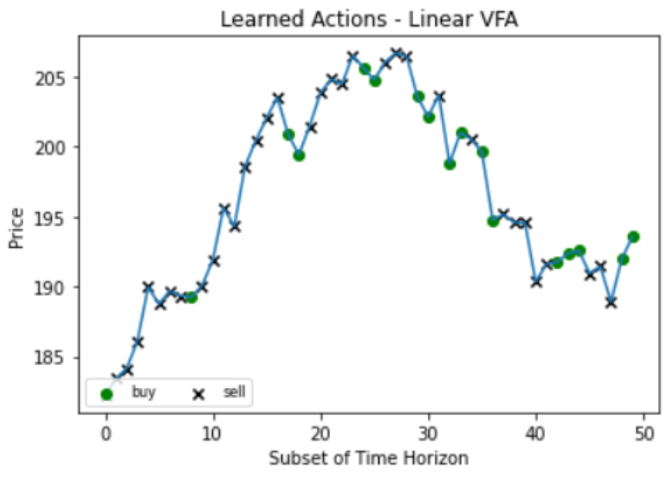

Fig. 9. HD Buying the Dip - Linear VFA

Once again, the agent has successfully learned the intended investment strategy with near perfection. It may even be possible to improve the performance on the agent with finer tuning of the features in the VFA.

Another interesting comparison would be to compare the profits in following each investing strategy. Here is a summary of the results.

TABLE III
INVESTMENT STRATEGY COMPARISON

Investment Strategy	AAPL	HD
Buy and Hold	\$75,240	\$198,640
Buy the Dip	\$81,889	\$226,710
Momentum Investing	\$84,073	\$227,251

According to the results, momentum investing clearly outperforms the other two strategies in this MDP model.

V. COMPARISON OF RESULTS

In this section, the results from the tabular methods and the VFA methods will be compared. The tabular methods produced results that were very similar to the buy-and-hold strategy. This makes sense because of the limitations made to the state space. For example, during the 10-year horizon there were 2515 trading days. This needed to be reduced further due to computational constraints of tabular methods. Thus, the agent could only execute one trade per month. This state space modification is equivalent to the agent holding the stock for the other 2389 trading days.

TABLE IV
AAPL - RESULTS COMPARISON

Method	Returns	Benchmark	Relative Performance
MC Tabular	\$76,029	\$75,240	1.04%
MC Linear VFA	\$84,073		10.51%

TABLE V
HD - RESULTS COMPARISON

Method	**Returns**	**Benchmark**	**Relative Performance**
MC Tabular	$200,319	$198,640	0.84%
MC Linear VFA	$227,251		12.59%

The results clearly show that the linear VFA method produced superior results than the tabular methods. If sufficient computational power was available, the tabular methods would have likely outperformed its counterpart. Instead, its results mirror the benchmark because the reduction of the state space essentially assigned an action of 'hold' to 2389 states.

VI. DISCUSSION

The goal of solving my MDP model was to find a policy that maximizes the total reward over some time horizon. This is significantly different from solving grid world based problems, where the agent must find a way to reach the goal state without much concern for the actual Q-values at each state. Here, the agent will always reach the end state irrespective of the sequence of actions taken. However, it needs to select actions that maximize the sum of the total rewards over the state space.

The only packages used were yfinance, pandas and numpy. I did not use any starter codes from any blogs or papers. I also developed my environment from scratch, basing it off the FinRL [1] paper.

Since this model is a significant oversimplification of how trading bots operate in the real world, it would be interesting to develop a more robust MDP that takes into account multiple market factors, with the ability to manage an entire portfolio of securities. It would be interesting to expand the action space to allow the agent to buy and sell various quantities of stocks, while taking into consideration the portfolio risk. In my experimentation, I used very basic techniques which are not very appropriate for this application. Most published research uses more sophisticated methods such as Deep Deterministic Policy Gradients [2], Deep Q-learning [3], and Recurrent Reinforcement Learning [3].

VII. CONCLUSION

Through experimentation, I have demonstrated that simple models like first-visit Monte Carlo can be used to get good results for this MDP. The tabular implementations were quite restrictive, yet still yielded policies that generated good results. These results were further improved by using a linear value function approximation for action-values. The most interesting result was that making slight modifications to the features significantly impacted the policies. These tweaks completely changed the incentive for the agent to take certain actions, with clear results. However, this MDP model requires significant improvements in order for it to be useful for real-world applications.

REFERENCES

[1] X.-Y. Liu, H. Yang, Q. Chen, R. Zhang, L. Yang, B. Xiao, and C. D. Wang, "Finrl: A deep reinforcement learning library for automated stock trading in quantitative finance," *arXiv preprint arXiv:2011.09607*, 2020.

[2] Z. Xiong, X.-Y. Liu, S. Zhong, H. Yang, and A. Walid, "Practical deep reinforcement learning approach for stock trading," *arXiv preprint arXiv:1811.07522*, 2018.

[3] J. E. Moody and M. Saffell, "Reinforcement learning for trading," *Advances in Neural Information Processing Systems*, pp. 917–923, 1999.