Optimizing Decision Making in Day Trading utilizing Reinforcement Learning

Apra Gupta and Andrew Rocchi
Northeastern University
(Dated: April 27, 2019)

In this paper, we explore decision making using Reinforcement Learning to optimize stock market trading decisions. We model day trading as a Markov Decision Process and explore Q-Learning as a way to optimize the decision of whether to buy, sell, or hold stocks. Instead of just predicting a price, we consider various environmental variables like the account balance, current portfolio, and prices from the last n-days of trading to arrive at the best action for a particular state.

I. DECISION MAKING IN STOCK MARKET TRADING

When trying to approach trading through AI, people think of predicting prices first. However, making money through the stock market boils down to making decisions on when to buy or sell your shares. Not only do you want to predict the price of a stock, you want to optimize how many stocks you should buy or sell to ensure you maximize your profits given the conditions you're trading in. So, it is not enough to just predict the price. Various factors in the environment like how much money you have, trading fees, network latency on your trading platform, how many stocks you own, etc. all play a crucial role in this decision. Simply price prediction is not enough to guarantee that you are selling the optimum amount of stock at the best time.In our project, we try to model a solution to the problem of stock trading that takes all these variables into account.

A. Day Trading as a Markov Decision Process

Our solution to this problem was to model the stock market as a Markov Decision Process. This would enable us to consider multiple environmental variables while figuring out the best course of action. We can then use reinforcement learning to decide what the best action to take is on any particular day given the various environment variables on that day and the stock prices from the last n days. So, in our project, we explore the effectiveness of modelling stock market day trading as a Markov Decision Process.

II. DATA

A. Input

1. *Linear Approximate Q-Learning*

Our initial data for Q-Learning is time series data where each day has an associated opening, closing, high, and low price. Initially, we used the 4 associated daily stock prices (opening, closing, high and low) of the S&P500 (Standard and Poor's 500, also known as GSPC) index. S&P 500 is an American stock market index based on the market caps of the 500 largest companies that have their stock listed on the NYSE or NASDAQ. Not only is this one of the most followed indices, it is also considered a pretty good overall representation of the performance of the American stock market. To train our algorithm, we used data starting from January 2^{nd} 2001 to December 31^{st} 2010. We evaluated our algorithm on data from the entire year of 2011. We obtained this data set from the website Quandl.

2. *Improved Polynomial Approximate Q-Learning*

It soon became apparent to us that the features we used in our initial algorithm were not on their own very useful in coming up with an effective trading strategy. So, we used a better data set with more expressive and relevant features in our improved Algorithm. This data set was structured in the same way as the last one except that along with the closing, opening, high and low price of very day, it contained 22 extra financial indicators derived from them that are very relevant in predicting the viability of day trading on the subsequent day. These indicators included Exponential Moving Averages, Stochastic Oscillators etc. Additionally, this data set also included a label which tells us whether or not it is profitable to day trade (i.e, buy stocks on the subsequent day's opening price and sell either some or all of them at it's closing price). The way in which all these features and labels are calculated is explained in the source of this data set[1]. We used the same index, S&P500, in this algorithm as well as a slightly smaller data set. We trained our algorithm on data from 2010-2015 and evaluated the algorithm on data from 2016.

B. Output

Initially, our algorithm would learn a policy that takes in the state representation of any given day and output the best action to take for that day given the various environmental variables (eg: buy 6 Apple shares) with the goal being to maximize profits for a period of trading. In our improved version of the algorithm, we still learn a policy, but it would output the best action to take on the subsequent day, with focus on intra-day trading (i.e, how many stocks to buy at the opening price of the next day and sell at the closing price of the next day).

III. TECHNICAL PROBLEM FORMULATION

1. *State*

We modeled a state essentially as a list of features. Initially, these features were the current account balance, the number of stocks that you currently own, a list of the past N closing prices of the stock, and also the trend of the data.This trend calculated the slope between the first and last day in the given N previous stock prices. This way, if there was a consistent slope in the data set, our model would be able to recognize that and account for it.
Later, with our new data set we modeled the state with more features (given in the data set) and also included the opening price of the next day as one of the known features. The closing price of the next day was not known to the state.

2. *Action*

Our action space includes buying, selling or holding. Additionally, we can decide how many stocks to buy or sell by adding an action per amount that we want. The algorithm was given a limited number of options for how many stocks it could buy or sell. For example, we could have an action space of [(buy, 1), (buy, 2), (sell, 1), (sell, 2), (hold)].
In our improved version, the actions were either hold, or buy X stocks at the opening price and sell them all at the closing price. Again, the algorithm was given a limited number of options for the number X.

3. *Reward*

We have experimented with several reward functions over the course of this project. The main two are:

1. Portfolio Gain: Holding Penalty + ((Account Balance + (Number of Owned Stocks * Current Stock Price)) - Starting Account Balance).

2. Profit: Stock Price - cost of most Recent Stock Bought.

With the 1st reward function, we have included a holding penalty in order to dissuade our model from just holding 100% of the time. For the second reward function, we keep track of the prices that we bought shares, and the reward is how much profit would be made by selling. In our improved algorithm, the reward was the difference between the closing price and the opening price on the subsequent day multiplied by the number of stocks traded.

learning curve.PNG

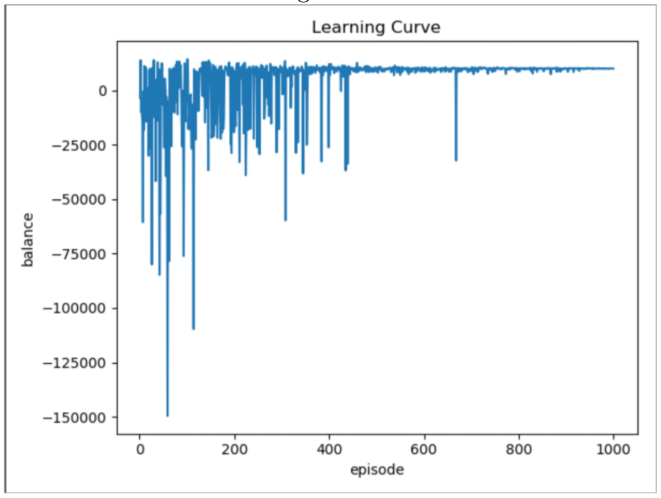

FIG. 1. Learning Curve

4. *Transition Function*

Given a state s (on day n), and action a carried out on day n we can find the next state s' (on day $n+1$) by using the closing price of day n+1. For example, if we bought 6 Apple stocks and the closing price on day n+1 was p, then the account balance on day n+1 will decrease by p*6 and the number of Apple stocks owned will increase by 6.In our improved algorithm, the account would simply increase or decrease by the reward (i.e. profit from that day).

IV. BASIC APPROACH: APPROXIMATE Q-LEARNING

A. Algorithm

We used approximate Q-learning[2], as learned in class, as our basic solution. So, we used the temporal difference error to adjust the weights on each feature of a state. Additionally, we used epsilon descent with our exploration policy.

B. Results

The learning curve looked pretty good at first glance. FIG 1. shows the account balance (on the y-axis) as a function of the episode (x-axis). Upon further analysis, we found that the policy we learned was just to hold no matter what. If we removed holding from the possible actions or imposed a very large holding penalty, we

Decay Decisions.png

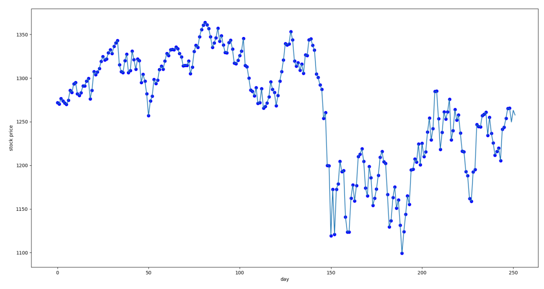

FIG. 2. Learned Actions

Curve Improved.png

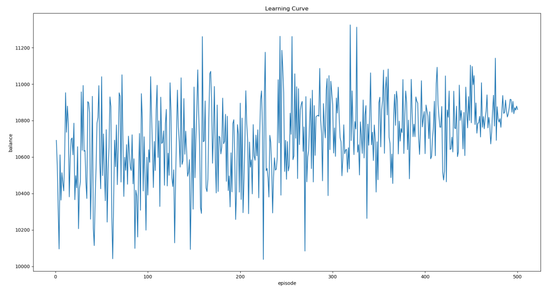

FIG. 3. Learning Curve

expected the policy learned would no longer just hold, but none of the policies without holding or with a high holding penalty were ever definitely better than the just holding policy, so we ended up holding the whole time anyways. FIG. 2 . shows the learned policy being evaluated. The blue circles indicate that at each day, the decision outputted is to hold.

C. Analysis

Upon tweaking various hyper-parameters and analyzing the weights, we realized that the expressive power of our Q-function was too weak. With the limited expressive power, the best function it could learn was indeed to not invest at all. There were too few features and they could only be combined linearly. To improve this, we decided to add new features that are not just linear combinations of the input data as well as to improve the expressive power of the Q-function.

V. NEW AND IMPROVED Q LEARNING

A. Algorithm

As alluded to before, in order to take full advantage of the better data set, we modeled the problem slightly differently. This time, we considered profit from day trading as the optimizing objective of our agent. We included all the features from the new data set into our state (not including the label because this would make the problem too easy) and our agent's job was to decide whether or not to buy stocks at the next day's opening price and sell them at the closing price and, if so, how many. Due to this simplification, we could eliminate the number of stocks held as a feature as we were no longer holding stocks across days. Another major improvement we made to the algorithm was that we added the support for polynomial transformations of our features. Our Q-function was rewritten to be a polynomial function of whatever degree we specified and this greatly improved

Test.png

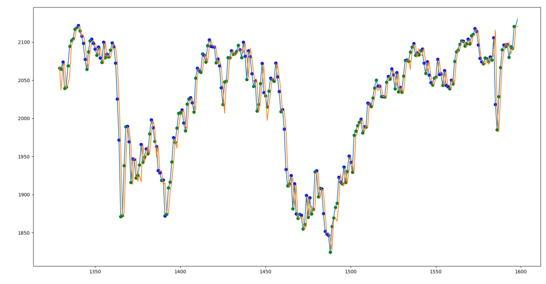

FIG. 4. Choices on Test

the expressive power of our model. Further, for the action space, instead of just having the option to only buy or a sell a single stock on any given day, we allowed for the model to trade multiple shares in a single day. We continued to use approximate Q-learning, meaning that we still used the temporal difference error to adjust the feature weights.

B. Results

With a polynomial degree of 1, we actually ended up making a consistent profit of around $800 on the test set! The learning curve looked promising as well, as shown in FIG. 3. Further, the choices as shown in FIG. 4 also looked quite promising. The algorithm seemed to choose to trade on days were the opening price was below the closing price, and hold on other days. The new features supplied to it proved to be effective!

C. Analysis

One difference between the data set that we were using for this improved Q-Learning algorithm versus the previous one was the trend of the data. While in the old data set there were ups, downs, and no clear trend, this new data set had a clear upward trend to it, with only minor dips in the stock price. This could potentially have skewed our results, since it would be much more difficult to lose money in this data set. However, the learning curve looks as though we are improving with each episode, so we believe that this is a superior method to our previous algorithm. Additionally, it is not as though we are buying as many stocks as possible on the first day and then waiting to sell them all, since we can only buy and sell on a single day time-frame. So, the benefit of the upward trend is mitigated a bit by this.

VI. SUPERVISED LEARNING + Q LEARNING

A. Algorithm

For this iteration, we had planned to accommodate supervised learning into our model by first building a classifier to predict the aforementioned labels (of whether or not it would be profitable to day-trade on the subsequent day) and then incorporate this as a feature into our state. This way, one of the features would tell us to some degree of certainty whether or not day trading on the next day was a good idea, and then our Q-Learning agent could refine that prediction based on the environmental variables as well as optimize the number of stocks to trade. We spent a lot of time trying to build a classifier but we found the training and test accuracy to be very low and did not want to spend any more time debugging or working on the classifier since we wanted to focus on the reinforcement learning aspect more. Instead, we decided to scaffold the problem by assuming we had a working classifier and built our reinforcement learning agent on top of it. So, we added the already known label to the features to see how much this would improve the algorithm's performance.

B. Results

This variation of the algorithm predictably does a lot better than the last one. It makes about $2000 on the test set. As is apparent in FIG.5, the learning curve is very promising. Further, the choices also seem to be really in line with what is expected to perform best, i.e, trade when the opening price is much less than the closing price. This is visible in FIG.6.

C. Analysis

Since this was essentially a labeled data-set, this is not surprising. This, however shows the premise that a layer of supervised learning can being can bring to Q learning for stock market data!

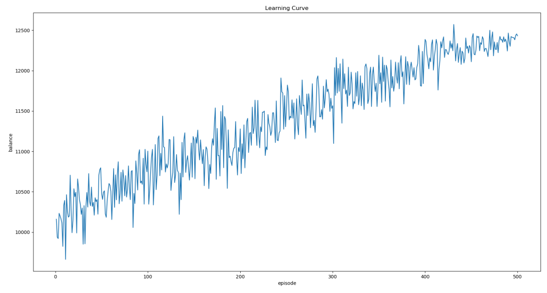

FIG. 5. Learning Curve

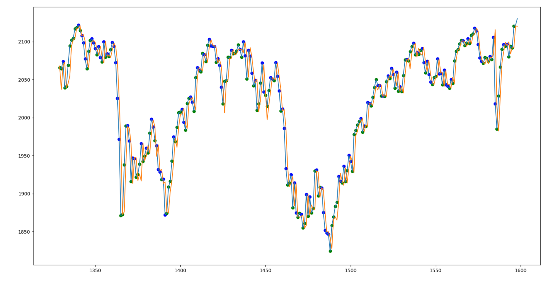

FIG. 6. Choices on Test

VII. DISCUSSION

A. Limitations

1. *Linear Formula*

There are many limitations to our approach that are yet to be overcome. Whether or not the features we ended up using are guaranteed to actually tell us much about the best course of action with any degree of certainty is still unclear. There are so many unpredictable outside factors that can come into play when trying to model the daily fluctuations of the stock market that we have not accounted for and that there is perhaps no known way to actually account for feasibly. There is no way, for example to feasibly consider every other participating agents behaviour in the stock marker, however, it plays a crucial role in selecting the best course of action. This could be one of the reasons why we found it so hard to build a classifier that predicts whether we should engage in day-trading on the next day. There are also other factors that are perhaps known but were not accounted for by out model like: trading fees, network latency on the trading platform, volume of stocks traded on a given day, signals from the news etc. Additionally, our model only considered trading one stock at a time. In reality, professional traders have many different stocks in their portfolio. Diversifying the portfolio considered by the agent may indeed lead to very different policies. Further, our last model assumes that we already have a reliable signal on whether or not we should engage in day trading on any given day. This is not true in the real world, and actually building such a classifier is an open problem. It is also possible that even a polynomial Q function is not expressive enough to be able to capture optimal policy. Perhaps a way forward would be to incorporate a deep neural network into the model as the Q function.

2. *Reward Function*

Another challenge we faced in utilizing Q-Learning to make a profit off of day trading was that it was difficult to find a proper reward function. One reward function which we used was to keep track of the last time we bought a stock, and have the reward for a state, if selling, be the difference of the selling price and the last buying price, which we would then pop off the stack if we ended up choosing to sell. This had some issues with it however, since it was unclear how we should reward buying or holding. Another reward function that we used was to calculate $(HoldingPenalty + ((AccountBalance + (NumberofStocksOwned * CurrentStockPrice)) - StartingAccountBalance))$. We used the holding penalty here since we wanted to dissuade the model from just holding, however the model still ended up holding most of the time. Additionally, the rest of the formula is just the difference of total value of what you own now and the value you started with. This was an attempt to make the model look at a larger time scale, since the previous reward function only looked at any given buy or sell profit, and not your total value.

3. *Hyper-parameters*

One common challenge in artificial intelligence is deciding on the values of hyper-parameters, and that certainly applied here. The hyper-parameters involved in Q-Learning include α, the learning rate, γ, the discount, and ϵ, the chance of choosing a random action, leading to more exploration. By adjusting our hyper-parameters, we first found that if we decreased the learning rate massively (0.00000000001 for 1000 episodes), we did not require normalization at all, since the weights never exceeded the value of any numbers in Python. Additionally, this led to our weights looking like a sort of cone, shown in FIG. 7 This, however, led to much slower learning, and gave a limit to the number of episodes we could train, so this idea was quickly discarded in favor of normalization. Next, we found that adjusting our discount had no obviously visible effect on training at all, which we hadn't expected, but given that our model always favored holding, does make some level of sense. Finally, changing our exploration with ϵ did change our training, but since we are always getting to the end result of always holding, the more we increased this hyper-parameter, the worse our evaluation performed (except for random times when it happened to perform well, but this wasn't consistent and was pure luck).

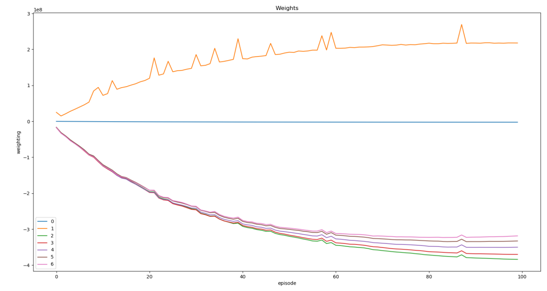

FIG. 7. Weights

B. Future Work

1. *Normalization*

Related to our work with hyper-parameters, we also had some issues with normalization. Since our weights were increasing to the point where they exceeded the size of a number in Python, we decided to normalize them. We flipped between 3 normalization functions, either using the weight divided by the sum of the weights, using $\frac{w_i - min}{max - min}$, or using $(2(\frac{w_i - min}{max - min}))$ - 1. The first method worked well as a normalization function, however the normalized weights ended up being rather small. For the second method, we got values normalized between 0 and 1. However, this neglected having negative weights, which is a problem for us since we may require negative weights if a feature has a negative correlation to us making a profit. To address this, we then used the third normalization function, which normalized between -1 and 1. While this fixed the prior issue, we noticed that for both the third and second functions the fewer features that we had, the closer the weights were to 1 and 0 or -1, and at least one of the weights would always be 1 and 0 or -1, unless the weights were the same in which case we got a divide by 0 error since (max - min) would equal 0. This is shown in FIG. 8.

C. Future Work

As mentioned before, incorporating a neural network into the Q-function seems like a reasonable next step. Another step would be building a good classifier that is actually able to predict, with some degree of certainty, whether a coming day is viable for day trading. Signals on whether to buy or sell could also be modeled by analyzing news sources and their opinions on the stock prices of a stock, potentially by using sentiment analysis

Decay Weights.png

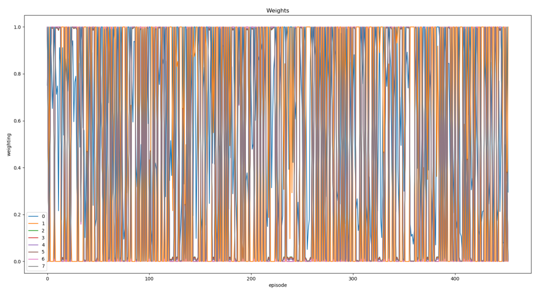

FIG. 8. Weights

to analyze data retrieved from web scraping. Instead of having only one stock in your portfolio, perhaps we could also explore seeing how things change when a more diversified portfolio is considered, as there could be trends within the data, such as one company's success affecting another company a few days later reliably.

D. General Experience

We spent most of our time writing and debugging code and testing different hyper-parameters. Reinforcement Learning has a lot of moving parts and hyper-parameters. We had a lot of fun learning about how the stock market works and have definitely gained a much greater appreciation for the difficulty of the problem, though we would have loved to make an accurate predictor and have made boatloads of money from it. Through our efforts in this project, we have gained experience with python as a language, and several libraries including numpy, pandas, and mathplotlib. Additionally, we have learned how to use Machine Learning with respect to our neural network as well, using the Keras API in Tensorflow and Sklearn. Finally, we have done plenty of research on how to choose hyper-parameters and reward functions, and have learned far more about these subjects than we have during lectures or homework assignments.

E. Advice/Insight

For attempting projects similar to this one in the future, we have a few suggestions:

- Unit test each function, and make sure to do so after each change. At first, we were running into many problems with our model making choices which we couldn't explain in any way, and changed around many of our functions to no avail, without realizing that our reward function was nonsensical and just returned the same value no matter what. With unit testing, we would have avoided that problem, and saved time debugging.
- Make small, modular functions which are well tested. When we first made our Q-Learning algorithm, we just wrote it in a script with few functions, and just had it run through the whole script without any custom objects to represent the actions or states. This made our program very difficult to test and build on.

- Think carefully about the features, reward function, and hyper-parameters which you use. Choosing these indiscriminately will lead to unreliable results at best, and completely wrong results at worst.

- Think about what is happening with each step in the reinforcement learning process. Similar to unit testing, thinking about what is happening during each step of the process can give you precious insight into why your model is behaving the way that it is, which makes debugging and improving your algorithm far easier.

[1] Bruni, Renato. Stock Market Index Data and indicators for Day Trading as a Binary Classification problem. Data in brief vol. 10 569-575. 29 Dec. 2016, doi:10.1016/j.dib.2016.12.044.
[2] https://github.com/d4nkfruit/AI_Project_Redone.