Deep Reinforcement Learning in the Stock Market

Eric Chan
Paul G. Allen School of
Computer Science and Engineering
University of Washington
Seattle, WA 98195
`yee96@cs.washington.edu`

Andrew Wei
Paul G. Allen School of
Computer Science and Engineering
University of Washington
Seattle, WA 98195
`nowei@cs.washington.edu`

Abstract

In this project, we explore the use of deep reinforcement learning in a stock market setting as a resource allocation problem. We then include predictive features in the observation space based on a recurrent model trained on stock prices. We showed that Reinforcement Learning algorithms can be applied in the realm of stock trading as we reframe the problem as a resource allocation problem, yielding results that proved to be better than that of an average investor.

1 Problem Statement

The financial market is a potentially interesting environment to perform research on using Deep Reinforcement Learning. One can imagine that the stock market is a multi-agent environment where each player attempts to outplay others by timing their entry into the market. Reinforcement Learning can be thought of as training an agent to maximize some reward metric defined by the user. With the recent advancement of deep learning technologies, the possibilities of its applications in solving reinforcement learning problems are endless. Following this reasoning, we would like to explore the possibilities of training a deep reinforcement learning agent to interact in the stock market environment, through the augmentation of state, so as to maximize its reward.

2 Approach

We start our project by recreating the setting used in [1] using Stable Baselines.

The setting in [1] has the following properties. Their observation space is made up of how much money they have, the closing price of each stock, and the amount of each stock (assets) they own. Then the action space is how much of each stock we want to buy or sell at each step (each day), limited by the amount of money we have and and the amount of each asset we own. On each day, we sell and then buy stock based on the closing prices of the previous day. The reward is the total amount of money in terms of assets owned and how much money is held at the end of each day.

This setting may not be the most realistic, as the closing price of a stock on some day and the opening price of the same stock on the next day can differ after the stock market closes. It may be worth mentioning that while we only care about the closing prices on each day in this project, we could have used the same approach with the changing prices while the stock market is open.

Stable Baselines[1] is an open-source set of implementations for different Reinforcement Learning algorithms, including implementations of Deep Reinforcement Learning algorithms. We decided to use Stable Baselines so that we could have more flexibility to try many algorithms. We mostly focus on DDPG, as that is the one we are the most well-versed with. We also performed some initial testing

[1]https://github.com/hill-a/stable-baselines

using some other Deep RL algorithms like TRPO, PPO2, and TD3; but did not perform extensive enough of an investigation to be conclusive about their results. We do acknowledge that they are all able to learn in this environment and achieve some sort of profit.

DDPG is an off-policy gradient-based approach to learning the Q function as well as the policy concurrently. In this experiments, we opted to use DDPG mainly because DDPG works well for continuous action spaces which is applicable to our stock trading use case. For example, we could imagine the the agent passing us floating points value for the amount of stocks that it wishes to purchase.

Lastly, we create predictions using a LSTM model that iteratively trains on available data to produce predictions. Note that by iterative training, we meant that to make predictions for day t we would train from day 0 to day $t-1$. This means that we would move data points from the test set, once that particular date has been predicted, into the training set. Then we would do 1-2 epoch update on the model to further update the model. We do this because financial data is finite, so including the actual data after making the prediction would give our model a better local view of trends as it goes forward than if we had not included it.

We include these price predictions in the observation space to see how well the agent can perform, given this additional information. Note that we had previously tried an approach with sentiment values as an additional feature in the LSTM model for better price predictions, but found that it does not work as well as expected; therefore, our final LSTM model simply takes in daily Open, High, Low, Close and Volume Traded to make predictions.

We train each agent on ten episodes of the training data, (from January 4th, 2016 to June 27th, 2018) and then test its performance on unseen test data (from June 28th, 2018 to December 30th, 2019). Since some algorithms depend on random initializations, we average its performance across many runs with different initializations and average its performance on each metric we care about.

In essence, we wanted to approach this task as a resource allocation problem. That is, we wanted to see how well the agent can allocate its resources to perform well in a stock market setting. This means managing its available balance and making the decisions based on what it owns at each step.

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
            $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$
        Update the target networks:
            $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
            $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
    end for
end for
```

Algorithm 1: DDPG algorithm, as described in [2]

3 Data

We obtain our data using Polygon, a financial data provider. We gathered daily open, high, low, close and volume traded for 15 tickers, namely: AAPL (Apple Inc), BAC (Bank of America), CMG (Chipotle Mexican Grill), DAL (Delta Airlines), GOOGLE (Google), JPM (JP Morgan), KO(Coca Cola), LUV (Southwest Airlines), MCD (McDonalds), MSFT (Microsoft), PEP (Pepsico), UAL (United Airlines Holding), V (Visa) and WFC (Wells Fargo). This data is collected for every day the stock market was open from the period of 2010-01-01 to 2019-12-30. We only use the data from 2016-01-04 to 2019-12-30 because there are data missing for some companies before 2016-01-04 and we wanted to make sure that the data was aligned.

Besides that, we also collected tweets of the 15 company tickers to perform sentiment analysis of the tickers, similar to the daily price data, we collected 10 years worth of tweets across the 15 companies tickers. We aimed to use the sentiment generated for the tweets to help make our price prediction model perform better which will in turn provide useful information for the agent to make decisions on.

4 Models

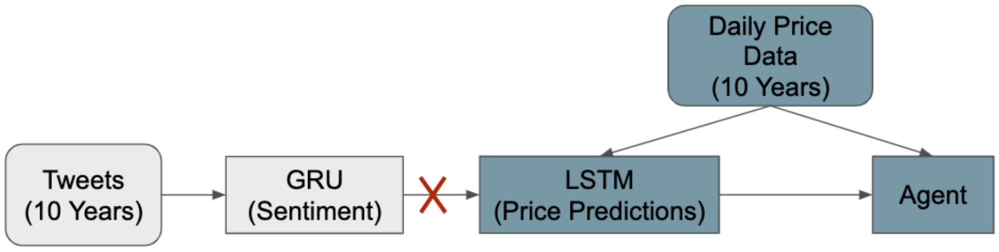

Figure 1: Model architecture

Our model architecture is presented above. Note that we tried using sentiment analysis results as an input feature into the LSTM model for price prediction. However, we realized through experimentation that it does not work as well as expected. Therefore, The final architecture only consists of the LSTM model and the agent trained using DDPG. Every step of the method indicates a single day of transitions being made by the agent. Every day, we will generate price predictions using the LSTM model and feed that, along with the daily closing price, into the environment to be accessed by the agent. The agent will then determine the allocations across stocks to either purchase or sell. We believe that adding the prediction features into our agent will allow it to make better decisions for allocating resources to maximize its profit.

The next model is the vanilla LSTM used for price predictions. We used a simple LSTM model with 2 layers and 32 hidden dimensions as well as 5 input dimensions to keep the model explainable. The 5 input dimensions corresponds to the open, high, low, close and volume traded for a ticker. We structured the LSTM model in such a way that each model corresponds to 1 company. Since we are tracking 15 companies, we have 15 models to train and maintain. Once the model made its predictions, it will then be passed into the agent.

5 Metrics and Evaluation

In order to evaluate our model performance, we used SPY (S&P 500 Exchange Traded Fund) as our baseline to compare against. SPY is designed to track the S&P 500 stock market index. Therefore, this symbol is a good indication of the overall market sentiment and health. We wanted to ask the question of what would happen if we invested all our initial capital into SPY vs. the Model. We then evaluate the performance across models that is trained with and without the prediction features to further show that models with prediction features performs better than the one without. To evaluate the model performance, we used the following metrics. We define S to be the sequence of days with where s_i is an element of S and indicates the cumulative assets of our model on ith day of the trading day. We define T_n to be the value of the asset invested in SPY on the last day assuming we invested at the same start time as the algorithm being tested on. Note that our initial investment capital for both SPY and the model is 10,000 dollars.

We plan to evaluate our agent based on a few metrics, first of which is profit. Profit is how much money we earned at the end in terms of asset value and balance minus the amount we started with (initial investment). The next is the drawdown ratio of the capital so that we can understand where the agent performs at its worst and what catalyst of states causes it to lose money.

Note that Drawdown measures the biggest percentage difference from the peak to a trough that is after the peak. This indicates how bad a model can perform across a set time period. The last metric is Alpha, which we define as how well our model performs against the baseline, which is the SPY symbols in our case.

$$\text{Profit} = s_n - s_0 \tag{1}$$

$$\text{Drawdown} = \max_{i \le j} \frac{s_i - s_j}{s_j} \tag{2}$$

$$\text{Alpha} = \frac{T_n - s_n}{10000} \tag{3}$$

6 Results

We present the performance of our price prediction model. Below is the table representing the results. Note that the results presented using iterative training performs better than the one without across every single ticker. We also compare it to a moving average baseline of the last ten days as a baseline performance of price prediction.

Stocks Symbols	SMA (10 days)	RMSE (w/o Iterative)	RMSE (w/ Iterative)
AAPL	3.80	11.36	4.60
BAC	0.63	0.52	0.47
CMG	17.70	13.43	11.76
DAL	1.34	0.96	0.87
FB	4.20	3.49	3.45
GOOGL	24.41	18.64	18.29
JPM	1.88	2.07	1.68
KO	0.69	0.96	0.52
LUV	1.29	1.02	0.91
MCD	2.22	7.45	2.37
MSFT	1.52	16.46	2.28
PEP	1.54	2.31	1.36
UAL	2.24	1.99	1.41
V	1.78	13.80	2.73
WFC	1.14	0.70	0.69

Table 1: Root mean square error of the prediction models with and without iterative training compared to the Simple Moving Average baseline.

Across the all the tickers in Table 1, notice that our LSTM model trained with the iterative method performs better than the regular, non-iterative methods. Our iterative model also generally performs better than the Simple Moving Average (SMA) of 10 days.

In Table 2, we present the results of our agent trained with and without the prediction features as an input into the algorithm. To generate these statistics, we run the trained agent across 100 iterations of 100 rollouts each, using a different initialization, and averaged the results across the time period to obtain the final results. Surprisingly, across the board, we see that the model without the prediction features performs slightly better than the model with the prediction features. This is obvious across metrics like Average Profit, Average Drawdown and Average Alpha. However, across the metrics, notice that they do not exhibit large differences. We suspect that this has to do with the model latching onto noises produced by the additional features. It could also be argued that this additional features does not provide any additional contextual information for the agent to make better decisions.

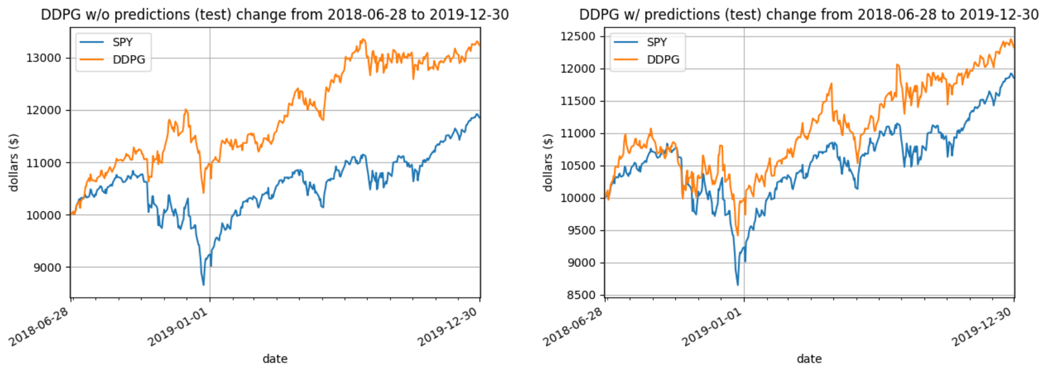

Figure 1: Notice that the left plot perform better than the right plot, indicating that the prediction values may not necessarily improve the model performance. Note that the initialization of DDPG is random, so the performance with or without predictions is not necessarily consistent.

Metric Statistics	w/o predictions	w/ predictions
avg. profit	\$4234.74	\$3833.60
std. dev. profit	\$1647.71	\$1586.84
max profit	\$7461.17	\$6807.26
min profit	\$1851.25	\$1809.90
average drawdown	15.44%	15.47%
std. dev. drawdown	2.94%	3.04%
max drawdown	23.37%	30.40%
min drawdown	9.86%	11.13%
average alpha	23.82%	19.80%
std. dev. alpha	16.48%	15.87%
max alpha	56.08%	49.54%
min alpha	-0.02%	-0.4%

Table 2: Results obtained after 100 iterations of 100 rollouts with different initializations for DDPG

7 Future Work

We initially focused only on one Deep Reinforcement Learning technique to perform the experiments. We believe that further exploration into the use of other Deep RL methods may provide more insights, as there may be other models that are better fit for the environment and interactions of stock trading.

We initially performed some experimentation with the inclusion of a sentiment analysis model in our predictions to ideally augment our belief of the state of each company through the use of scraped financial tweets. We later learned that the sentiment model did not generalize to the setting of financial tweets, as it was trained on general tweets for general sentiment. Thus, it may also be interesting to explore the use of sentiment in making decisions or predictions.

We are also looking into including more domain-specific knowledge into the model through adding additional features that would encode things like volatility through volume analysis.

If we weren't looking at the problem as a resource allocation problem, we could also break our RL model into smaller agents. This makes it so that they can be in charge of individual actions instead of having to learn how all the stock prices and assets interact with each other and can focus on more specialized information. This can also be parallelized and the individual models will be smaller, allowing for potentially faster training and a more powerful model overall.

Lastly, we believe that our model can be easily extended to the environment of intra-day trading, i.e. making decisions to buy or sell within the day. We believe that this is our next immediate goal, as doing so will provide much more information than the closing price of each day and allow us to train our agents on more data. As an extension to this, we would be able to get it to perform decisions in real time. We acknowledge that there would be some delay in taking actions due to the need of monitoring and processing data streams.

8 Conclusion

In this project, we took an applied approach to solving the problem of resource allocation. Then, we used DDPG an the algorithm to train an agent to solve this problem.

Next, we used iterative training as a way to improve out LSTM model price prediction performance and showed that it yielded decent results. We then used this feature to augment the state of our agent. However, we found that the price prediction features do not work very well as an additional feature for the agent to train on. This could be due to the fact that out model is latching onto noises in the dataset. However, we still strongly believe that this prediction feature can be utilized in some other form to provide contextual information which can further improve the efficiency of the agent in terms of making decisions regarding resource allocation.

We believe that while our attempts at improving the agent's performance through state augmentation did not perform as we expected, there are still possibilities to reframe our problem such that it would be able to perform better.

References

[1] Z. Xiong, X.-Y. Liu, S. Zhong, H. Yang, and A. Walid, "Practical deep reinforcement learning approach for stock trading," 2018.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.