Trading Stock with Deep Reinforcement Learning

Yixiao Song, Zhongjie Li, Stephen Guan Feng Yang

Stanford, California, United States

1. Task definition

The stock market has largely been a mystery to many investors. Few have been able to consistently and reliably predict its movements. We hope to solve this challenge by building an AI agent that can accurately predict stock trends. We hope to apply several variants of the technique of Reinforcement Learning, including deep Q learning, and double Q learning, to build a stock trading agent. We will compare the total rewards obtained by picking actions according to each of the generated policies.

2. Literature Review

Stock market prediction have always been one of the most interesting topic for artificial intelligence. As it is impacted by many different factors, it's very hard to predict, no matter if it's ”fundamental analysis” or ”technical analysis”.

Malkiel [1] concluded that no investment system can consistently yield average returns exceeding the average returns of a market as a whole after testing these systems on 40 years' data. Chen et al built a LSTM model to predict stock price in Chinese stock market and it effectively improved the return. [2] As social media become more and more popular, the twitter data can be a good source of information for stock market prediction.

In 2011, Bollen et al implemented a Self-Organizing Fuzzy Neural Network based model to predict stock market trend based on twitter data.[3] Neuneier proposed to use Q learning to help allocate assets for stock trading. [4] Lee et al proposed a daily stock trading policy based on Q learning, which showed promising results in Korean stock market. [5]

3. Infrastructure

3.1. Model

We will be running deep Q-learning with additional feature sets, which is a model-free approach. We'll define the following to model the market:

State: (last ten days' stock price, last ten days' index price, cash amount, stock amount)

Action: Buy or Sell

$Q_{opt} : maxQ(s, a; w)$

γ : interest rate

Reward: increase of asset(based on current cash and stock holding)

Cost: Stock transaction fee, time delay (not all transaction can be fulfilled immediately)

Output: final reward value (Cash + # Stock*price per stock)

Feature set:

LSTM time series forecasting

Sentiment analysis from social media data

Index (SP 500, Dow, Nasdaq etc)

3.2. Dataset

The dataset we used for our results is the daily(Open and Close) stock price history of SAP from 1/1/2016 to 10/1/2017. We are training our data from 1/1/2016 to 1/1/2017, and then run prediction from 1/1/2017 to 10/1/2017.

We also have about 13 million tweets from twitter tracked by the companys name that we used to do sentiment analysis, which we hope to incorporate into features of some of the Q learning variants that we do.

For our current model, we introduced a $ 2 cost with each action taken. We defined the total reward to be the total cash at the end of the test period, starting initially with $1000.

Stock price and indexes data from Panda data reader:

https://pandas-datareader.readthedocs.io/en/latest/

Twitter data from live tweets adaptor, tracking single company's name over 2 years period around 12 M+ data:

Data structure: id,created(time stamp),text,language, username,replyUser,retweeteduser

Figure 1: Some sample twitters with positive and negative sentiment for SAP's stock

3.3. Evaluation metric

Total asset at the end of test period compared to baseline and oracle.

4. Approach

4.1. Baseline

For the baseline of our project, we are doing a greedy algorithm as follows: if we have shares in hand, sell when price rises, and if not, we buy when price falls. (compare today's price with yesterday's). With an initial investment of 1000 into the market, between 2017-1-1 and 2017-10-1, our total cash in the end was 1046.09.

4.2. Oracle

For the Oracle of our project, we will do a linear search over the period 2017-1-1 to current date, which will find the optimal buying and selling date between the period. We will compare the data output after learning against this data to compute the loss. 41,698.21 USD is the total reward after running the linear search.

4.3. Stock price prediction

To predict the future stock price, we implemented two methods, including linear regression and deep neural network.

4.3.1. Problem setup

We used SAP stock prices between 2015 and 2017 as training dataset and the prices beyond 2017 as eval dataset. The input is the last 10 days' stock prices and the prediction is next 10 days' stock prices. The cost is calculated as following:
$cost = tf.reduce_sum(tf.pow(prediction - true_prices, 2))/total\_input\_qty$

4.3.2. Linear regression model

The most straightforward method to predict stock prices is linear regression.
$Prediction = W * X + b$
With 1500 iterations of training, the training loss is decreased from 49766 to 83. And the eval loss is 53. Following figure showed the prediction (in blue) vs the ground truth (in red).

4.3.3. Fully connected neural network model

We also constructed a 4 layer neural network to predict future stock prices. The network is structured with 4 hidden layers with size of 1024, 512, 256, 128, respectively. And the final output size is 10.

With 1500 iterations of training, the training loss is decreased from 102 to 46. The eval loss is 38.

4.3.4. LSTM neural network model

Considering stock prices are sequential data, the LSTM model should work better for that. The model setup is shown as following.

Following figure showed the comparison between prediction (in blue) and ground truth (in red). We can see that the predicted data overall follows the trend of the ground truth data.

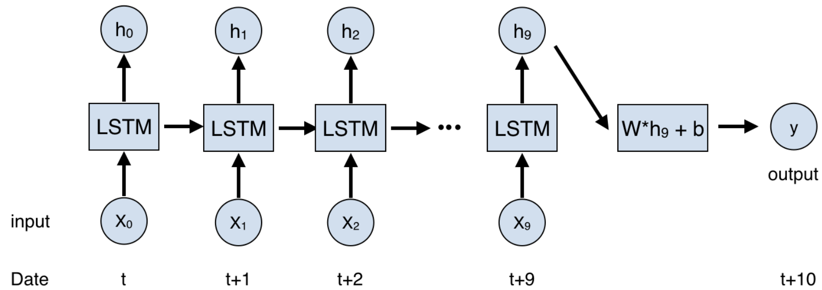

Figure 2: LSTM model structure

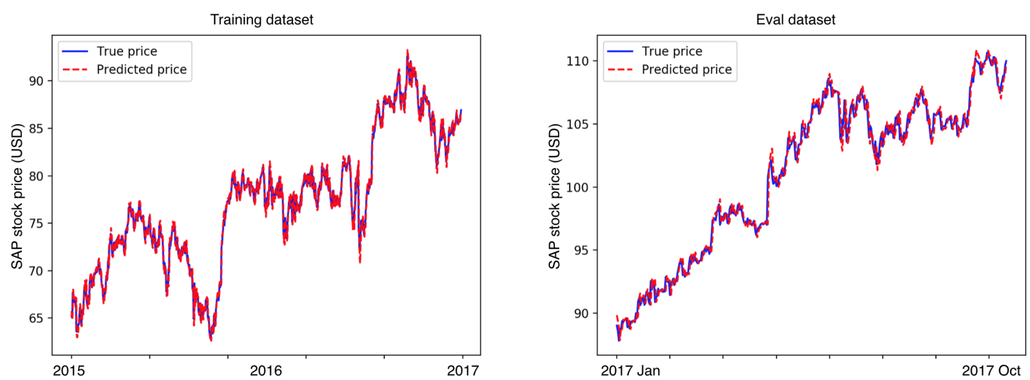

Figure 3: Comparison between ground truth and prediction, left: training data, right: eval data

4.4. Sentiment classification

Social media data has produced a crescent interest in the task of sentiment analysis. In this task, we will need process 13M social media data from Twitter, which tracking by single company name. However, there are few challenges we found out during the implementation.

1.From the dataset, there are over 60 different languages: en: 4718581, tr: 4034451,und: 1053715, pt: 536049 etc. In order to process all the languages, we will need pre-trained word vectors or labeled source training set.

2.Tweet from twitter is limited to 140 characters. The limitations bring a challenge to the regular bag-of-words model by providing very few contextual data.(n-grams only have 60% accuracy)

Based on the scope of the project and timeline, leverage NLP APIs to achieve the task is more advisable.

We are using google NLP API(https://cloud.google.com/natural-language/), Sentiment Analysis to process tweet of supported languages (English, Spanish, Japanese, Chinese (Simplified and Traditional), French, German, Italian, Korean and Portuguese.)

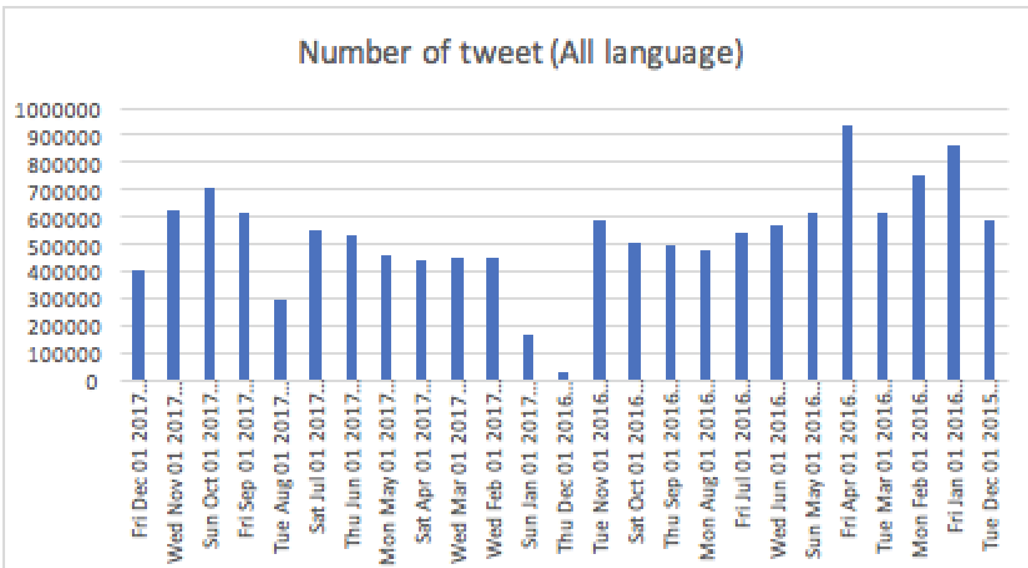

Figure 4: Number of tweets on 'SAP' by date

For this task, we build a scheduler agent to load data from social media table and process through sentiment API and store the result in the sentiment column.

Sentiment score for each tweet range from -1.0(Strong Negative) to 1.0(Strong Positive) and we are using sum of the tweets sentiment as well as average of the sentiment for our feature:
Result chart

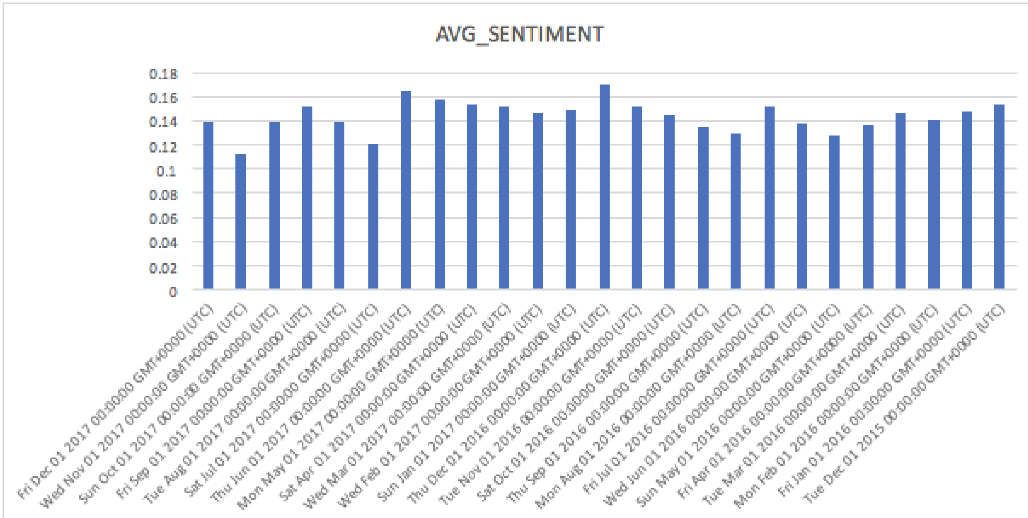

Figure 5: The average sentiment of 'SAP'

4.5. Q-learning

We've defined the model we are going to use for the agent above. We are interested in finding the maximum reward for a stock, so for each state we just want to find the maximum expected utility, $V_\pi(s)$, and it's corresponding action. This would in turn give us a policy π to predict future actions.

Due to the huge state space of the stock market(infinite), we need to use function approximation to estimate Q_{opt} for unseen (state, action) pair. Instead of having a simple linear $\phi(x) \cdot w$, we decided that our Q_{opt} will be approximated using a neural network that is 3 layers (hence deep Q learning), with the lower most layer outputting actions { "buy", "sell", "hold"}. The equation is as follows:
$Q(s, a) \leftarrow r + \gamma Q(s', argmax_a(Q(s', a)))$

The problem with the above approach is that Q learning by itself is noisy because it picks the highest estimated Q(s',a') to update the current estimated Q(s,a), which when next updated will further amplify the affect of the estimated Q(s',a'), whether or not it is actually the Q_{opt}. Therefore, we introduce a second Q as described in Hado van Hasselt's Double Q-Learing[6], as follows:
$Q_1(s, a) \leftarrow r + \gamma Q_2(s', argmax_a(Q_1(s', a)))$

and
$Q_2(s,a) \leftarrow r + \gamma Q_1(s', argmax_a(Q_2(s',a)))$
which van Hasselt proved to solve the problem described.

To address the exploration vs the exploitation problem, we don't fully rely on the estimated Q values, and use a ϵ-greedy approach to make sure we see as many states-action pairs as possible. (Which we label in our graph RAM(randomized Agent memory).We also added in one of our models optimizations PER(Prioritized Action Replay) to our Q learning Algorithm. The main idea behind Prioritized Action Replay is that we would pick cached replays that are worse fits of our model than better fits, so our model will learn the most from those replays.

Result we have comparing all the implementations:

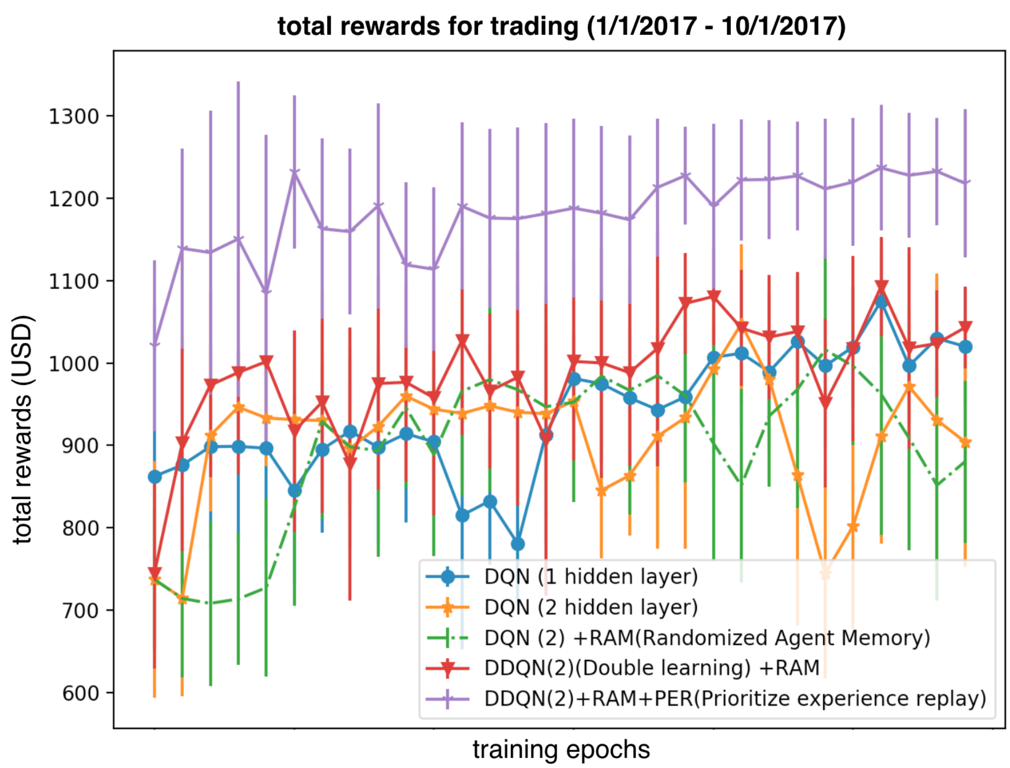

Figure 6: The comparison of several different Q learning algorithms

5. Error Analysis

1. For all the Q algorithms we implement during learning phase

DQN(1)	DQN(2)	DQN(2)_RAM	DDQN(2)_RAM	DDQN(2)_ADV2
100%	97%	96%	105%	126%

Figure 7: The average profit for different Q learning

we found the DQN with single layer have more robust performance in average reward,and DQN with two layer become less stable. This could because of the unappropriated network size we had for DQN with two layer(64+128).

2. Mean square error loss function could lead to network weights change substantially when the predict value have large bias from estimated value. We found Huber loss is more robust and less sensitive to outliers. Which will help the performance become more stable.

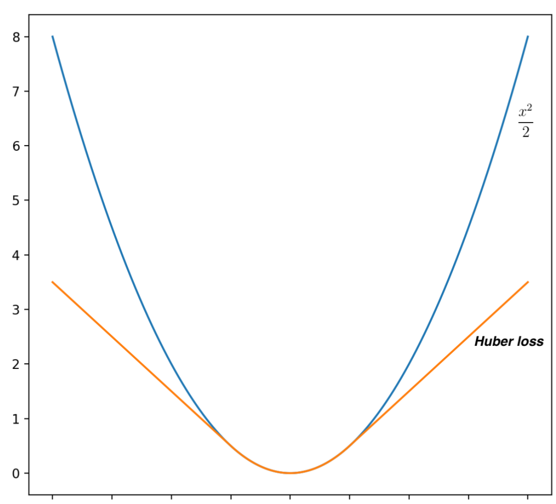

Figure 8: Huber Loss vs MSE

3.We tried to use longer time to learn our Q policy(from 2000-2017), but the performance is very unstable event in the learning phase(range from 10,000+ to less than 500 base on 1000 initial investment and 2$ transaction cost). The possible reason could be the memory limitation(We can not store all the possible states for very large data set).

4. Social media data keyword tracking to the unrelated tweet. Because we are using single key word(SAP) to pull data from twitter. And this keyword could be use differently in other language or irrelevant words. This could have bias on the sentiment result.

6. Reference

[1] B. G. Malkiel, A random walk down Wall Street: including a life-cycle guide to personal investing, WW Norton & Company, 1999.

[2] K. Chen, Y. Zhou, F. Dai, A lstm-based method for stock returns prediction: A case study of china stock market, in: Big Data (Big Data), 2015 IEEE International Conference on, IEEE, pp. 2823–2824.

[3] J. Bollen, H. Mao, X. Zeng, Twitter mood predicts the stock market, Journal of computational science 2 (2011) 1–8.

[4] R. Neuneier, Enhancing q-learning for optimal asset allocation, in: Advances in neural information processing systems, pp. 936–942.

[5] J. W. Lee, J. Park, O. Jangmin, J. Lee, E. Hong, A multiagent approach to *q*-learning for daily stock trading, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 37 (2007) 864–877.

[6] H. V. Hasselt, Double q-learning, in: J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, A. Culotta (Eds.), Advances in Neural Information Processing Systems 23, Curran Associates, Inc., 2010, pp. 2613–2621.