Machine Learning Engineer Nanodegree

Capstone Proposal

S. A. M. Saddam Chowdhury

1 Domain Background

An automated trading system that makes investment decisions in a fully automatized way and generates constant profit from the financial market is lucrative for every market practitioner. Reinforcement Learning (RL) is a branch of Machine Learning (ML) that allows to find an optimal strategy for a sequential decision problem by directly interacting with the environment. In this project, I will develop an automated trading algorithm based on RL. The project is inspired by Udacity's "Machine Learning for Trading" course [1].

2 Problem Statement

In RL based stock trading, the agent is trained to interact with the market to achieve some intrinsic goal (e.g., 20% expected return). Common interactions are to observe the latest and historical financial data or submit new orders to the exchange, etc. Here I will construct a Double Deep Q Network (combination of a Double Q-learning system with a deep neural network) aka DDQN that will operate on multiple stocks and perform trading tasks by taking one of three possible actions (buy, sell or hold) at the close of each trading day. In stock trading, a state is a combination of portfolio and market inputs and represented as an array of # of stocks owned, current stock prices and cash in hand. In any state, based on the current and historical market data, the agent chooses an action which changes the portfolio structure as well as some market inputs. As a result, a new state is obtained. The goal of the agent is to learn, by trial-and-error, which action maximizes its long-run rewards. In Double Q-learning, the selection is decoupled from the evaluation [2,3,4]. Since the input space can be massively large, I will use a Deep Neural Network to approximate the $Q(s, a)$ function through backward propagation where 's' is the state and 'a' is the optimal action associated with that state to maximize its returns over the lifetime of the episode. Over multiple iterations, the $Q(s, a)$ function converges to find the optimal action in every possible state it has explored.

3 Datasets and Inputs

The dataset that will be used in this project include 8 years of daily stock data (ranging from 12/01/2010 to 11/30/2018) of different stocks. This data will be split into training and testing sets by 0.5 which will then be used to create the trading environment for training and testing separately. The data can be downloaded using Yahoo Finance API. Data can also be fetched from other sources such as IEX and Morningstar API. By increasing the length of the training window (data) , better performance can be achieved. However, with a longer training window, it takes too long for the computer to process the information. Using a powerful machine, more data can be fed in to the model.

Description of the stock data:

- **Open** - The price of the share when the stock market opens in the morning for trading.
- **High** - The highest value of the share for that day over the course of the trading day.
- **Low** - The lowest value of the share for that day over the course of the trading day.
- **Close** - The final price of the share for that day at the close of the trading day.
- **Volume** - The number of share traded for that day.
- **Adj Close** - An adjusted closing price is a stock's closing price on any given day of trading that has been amended to include any distributions and corporate actions (such as stock splits, dividends/distributions and rights offerings) that occurred at any time before the next day's open. The adjusted closing price is often used when examining historical returns or performing a detailed analysis of historical returns.

Only the daily 'Adj Close' price will be used in this study, though other features can be easily incorporated into the model. For n-dimensional stocks, I will create an n-dimensional array of data containing only the 'Adj Close' column from the original dataset. The trading environment will use this dataset to calculate the daily profit and loss.

4 Solution Statement

Several reports can be found on automated trading based on Q-learning/DQN. One problem with the Q-learning/DQN approach is that the agent tends to overestimate the Q function value, due to the max in the formula used to set targets:

$$Q(s,a) \rightarrow r + \gamma max_a Q(s',a)$$

A solution to this problem was proposed by *Hado van Hasselt (2010) [3]* and termed 'Double Learning'. The idea of Double Q-learning is to reduce overestimations by decomposing the *max* operation in the target into action selection and action evaluation. In this new algorithm, two Q functions – Q_1 and Q_2 – are independently learned. One function is then used to determine the maximizing action and second to estimate its value. Either Q_1 or Q_2 is updated randomly with a formula:

$$Q_1(s,a) \rightarrow r + \gamma Q_2(s', argmax_a Q_1(s',a))$$

or,

$$Q_2(s,a) \rightarrow r + \gamma Q_1(s', argmax_a Q_2(s',a))$$

It was proven that by decoupling the maximizing action from its value in this way, one can indeed eliminate the maximization bias. It has also been reported that DDQN approach exhibits better stability, less volatility and faster convergence in terms of model performance compared to its DQN counterpart ([3, 5, 6]). Therefore DDQN algorithm will be employed in this project to address the performance issues of the existing models.

5 Benchmark Model

Several interesting reports/articles can be found on RL based automated trading ([7,8,9,10]), but I am particularly interested in this work [5]. However, the author only considered a single stock and did not take into account any initial investment. In this project, I will develop a multi-stock portfolio with an initial investment of $10,000$. In another example ([11]), a different author considered the DQN approach to tackle a similar problem. In this project, I will develop/implement a DDQN based trading agent. It will be interesting to see if DDQN algorithm improves the model performance in terms of speed (faster convergence) and stability (i.e., high volatility in portfolio values reported in [11]). It will also be interesting to find how the model performs in the case of multi-stock portfolio as compared to the single-stock one reported in [5].

6 Evaluation Metrics

A reward is a scalar feedback signal that indicates how well an agent is doing at a specific step. The agent's job is to maximize the cumulative reward. There are several possible reward functions we can pick from. An obvious one is the instantaneous reward of daily profit. The net profit from that trade can be positive or negative. That's the reward signal. As the agent maximizes the total cumulative reward, it learns to trade profitably. The reward function is calculated as

$$R = v_t - v_{t-1},$$

where v_t = portfolio value (cash_in_hand + stocks_owned * stock_price) at the t-th day.

Generally the performance of an RL agent is assessed by the total cumulative rewards over all episodes as the model converges to a stable solution. Therefore, higher the cumulative rewards and less volatile the behaviour of the portfolio values, the better. In this project, I will consider an initial investment of $10,000$ for trading on 3 stocks (MSFT, AMZN and IBM). The goal is to maximize profit with expected return of at least 20%.

7 Project Design

I will follow the following sequence of steps:

1. Acquire proper understanding of the financial data to be used and its features, and carry out initial data analysis and statistical comparison among various stock data. For example, before investing in a stock, it is important to analyze the market trend, stock volatility and correlation among various stocks if they are positively/negatively correlated.

2. Preprocess the data and check date alignment of the stock data (some stocks may not have prices listed for all the days the stock markets are open).

3. Reproduce and analyze the existing results using the newly developed code. This will ensure the accuracy of the code as well as the consistency of the results to be produced.

4. Study existing algorithms, identify the problems with the existing models and select an appropriate algorithm which will likely to improve the performance. In this project, I will employ DDQN algorithm to develop the model (see discussions above).

5. Fine-tune the hyperparameters to further optimize performance.

6. Test the model with various different inputs to verify the robustness of the model.

References

[1] https://www.udacity.com/course/machine-learning-for-trading–ud501

[2] "Reinforcement Learning: An Introduction", 2nd Edition, Richard S. Sutton and Andrew G. Barto

[3] https://papers.nips.cc/paper/3964-double-q-learning.pdf

[4] https://www.nature.com/articles/nature14236

[5] https://www.kaggle.com/itoeiji/deep-reinforcement-learning-on-stock-data
[6] http://torch.ch/blog/2016/04/30/dueling_dqn.html
[7] https://medium.com/@gaurav1086/machine-learning-for-algorithmic-trading-f79201c8bac6
[8] https://lucenaresearch.com/deep-reinforcement-learning/
[9] http://www.wildml.com/2018/02/introduction-to-learning-to-trade-with-reinforcement-learning/
[10] https://www.youtube.com/watch?v=rRssY6FrTvU&t=469s
[11] https://shuaiw.github.io/2018/02/11/teach-machine-to-trade.html