**SmartTrader**

ISS SLS PROJECT REPORT

ONN WEI CHENG A0092201X

YANG JIESHEN A0003901Y

NIRAV JANAK PARIKH A0213573J

ISA-PM-SLS-2021-01-09-IS02PT

Image from: https://static.vecteezy.com/system/resources/previews/002/062/633/original/business-candle-stick-graph-chart-of-stock-market-investment-trading-on-white-background-design-bullish-point-trend-of-graph-illustration-vector.jpg

**TABLE OF CONTENTS**

[1. Problem Statement 2](#_Toc72072460)

[1.1. PRoblem description 2](#_Toc72072461)

[1.2. approach to solution 2](#_Toc72072462)

[2. model overview and implementation 3](#_Toc72072463)

[2.1. state 3](#_Toc72072464)

[2.2. reward 3](#_Toc72072465)

[2.3. actions 3](#_Toc72072466)

[3. model evaluation 4](#_Toc72072467)

[3.1. features used for evaluation 4](#_Toc72072468)

[3.2. phase 1 4](#_Toc72072469)

[3.3. phase 2 4](#_Toc72072470)

[3.4. phase 3 4](#_Toc72072471)

[4. conclusion 5](#_Toc72072472)

**APPENDICES**

APPENDIX I:

1. Problem Statement

* 1. PRoblem description

A large volume of financial instruments are traded daily on exchanges and a growing majority of these are traded using automated trading algorithms. An automated trading strategy that maximizes profit is highly desirable for mutual funds and hedge funds.

These trading algos can be broadly categorized into:

1. Passive / Fundamental Trading which is primarily a long-term strategy based on market and security fundamentals or passively benchmark index trading.
2. Active / Price Action Trading which is based on buying and selling securities based on short-term movements to profit from the price movements and relies on historical prices (open, high, low, and close) to make trading decisions.

This project will focus on the Price Action Trading which is driven on the characteristics of a security’s price movements. Since it ignores the more subjective fundamental factors and focuses solely on recent and past price data, the Price Action Trading Strategy is more conducive to Machine Learning.

There are essentially two contrasting approaches to developing price trading strategies:

* 1. Model based: This approach attempts to create a mathematical model of the market thru representative variables (eg mean price, corelations) to create a simplified representation of the true complex market model. Some examples of this are trend-following, mean reversion, arbitrage strategies etc
  2. Model free: Here there is no attempt to model the market, rather it looks at price patterns and attempt to fit an algorithm to it. There is no attempt to produce a causal analysis or explanation – just an attempt to identify patterns that will repeat in the future. Examples here are Technical analysis, charting, candle patterns etc

This project will focus on the Model free approach and leverage **Reinforcement Learning** (RL) to develop an automated trading agent (**Smart Trader**) driven solely on historical data and derived technical features.

* 1. approach to solution

While RL has traditionally proved its capabilities in learning to play games, RL presents a unique opportunity to model the complexities of the trading strategy as a game in which an agent can be trained to maximize its total reward from the market.

Trading is a continuous strategy in which decisions are taken sequentially and which collectively culminate into a profit or loss. As such, there are no specific label associated with any historical data and this is a major problem for traditional Supervised Learning (SL) approaches. On the other hand, RL can use feedback from its own action and experiences ie the rewards returned by the stock market to learn an optimal trading strategy.

RL also has different learning goals from SL. While SL learns to make the best predictions (Classification or Regression), RL learns a policy for actions that would maximize its long-term cumulative reward, which is goal of trading.

Trading is a continuous activity and has no defined endpoint. Also trading is a partially observable environment as we do not have complete information about the traders in the market. Since we have a Partially Observable Markov Decision Processes (POMDP) and we don’t know the reward function and transition probability, we use model-free reinforcement learning which is **Q-Learning**.

Q-learning is a model-free reinforcement learning algorithm. It is a value-based method that supplies information (exploit) to an agent for the next trading action (buy, hold, sell). It is an off-policy algorithm as the q-learning function learns from occasional random actions (explore) that are outside the current policy.

**Deep Q-Learning** approximates the Q-value with a neural network as a function approximator.

Q-value refers to the action quality and is stored in Q-table with dimensions [state, action]. The Q-value is the maximum discounted future reward when an action a is performed in a state s. The objective of Q-learning is to learn the optimal value of the Q-table that will provide the maximum reward at the end of the n number of training cycles or iterations. It functions well without the reward functions and state transition probabilities.

At run-time, the agent will evaluate rate each and every action from the current state and select the one with the maximum Q-value

1. model overview and implementation

The analysis and work done in this project aims to improve on an existing Deep Q-Network reinforcement learning model which is available on Kaggle: <https://www.kaggle.com/itoeiji/deep-reinforcement-learning-on-stock-data/notebook>

The model divides the data set into the training dataset and the test dataset. The agent learns using the training dataset and is used to predict buy/sell/hold actions to maximize profits.

A deep neural network is used to predict the action value based on input states. Due to the nature of stock markets, prices are somewhat correlated sequentially. Hence, an experience relay is deployed to stabilize the network. The agent uses a memory to store batches of historical data which is parsed in randomly to train the neural network. The target Q network is updated periodically to further reduce correlation.

The hyper-perimeters are listed in Table xx, and the structure of the neural network is displayed in Table xx

* 1. state

A **state** is the description on the current environment that the agent finds itself. The details on this state are provided to the agent by the environment. It is important that this state has the [Markov Property](https://en.wikipedia.org/wiki/Markov_property) ie the current state information has sufficient and necessary information for the agent to accurately predict expected next rewards and next states given an action, without the need for any additional information.

In our trading model, our OHLC daily price data is a time series and is not stationary ie there is a trend embedded within the time-series. We make the daily OHLC price data by calculating the daily returns ie today’s price – yesterday’s price. This makes the returns data stationary and this is what we use in our model to describe current state.

Furthermore, given the inherent trends in the price data, a single day of price returns is not sufficient to describe the complete state of the market. Hence we use a sliding window of n days of historical returns data (defined by parameter historical\_n = 90) in the state description to provide the agent the full view of market movements in the recent past.

Thus the full state returned by the environment on every step is the today’s return + returns for the historical\_n days. This state is input into the NN network to model the Q-value for the state-action value.

* 1. ACTIONS

Actions is the set of all possible moves the agent can make from a given state. For our trading agent, there are only three permissible action: BUY, HOLD or SELL.

In order to affect the off-policy learning the model employs an Epsilon Greedy Policy to find the balance between the model’s Explore vs Exploit behaviour. The model defines a starting epsilon value (epsilon = 0.9) which starts decreasing by a pre-defined value (epsilon\_decrease = 1e-3) after a predefined number of step (start\_reduce\_epsilon = 200) till it reaches the specified minimum value (epsilon\_min = 0.1). This allows the model to start with a high exploration rate and hence learn quickly during the initial steps and gradually reduce exploration and increase exploitation as the model starts learning the optimal policy.

For each of the actions, the environment takes a distinct set of steps to calculate the current rewards. These Reward calculations will be described in the next section. The state returned after any action is similar and as described in the previous section ie the environment returns the current returns + historical\_n returns.

* 1. REWARDS

A **reward** is the feedback provided by the environment that measure the success or failure of an agent’s actions in a given state. For our environment, the rewards are calculated distinctly for each trading action (Buy, Hold, Sell).

The reward for each action is primarily the returns generated for each action. Further more, we have introduced the concepts of TRANSACTION COST and HOLDING COST to account for the real-life trading costs.

* Transaction cost = the cost of executing a trade and is deducted from the rewards of the Buy action. This is a % cost of the $$ stock bought at the Buy action.
* Holding Cost = Financing cost of holding a position over a Hold action step and is deducted from the reward of the Hold action. This is a % cost of the $$ position that is Held over that step.

Hence the rewards from the environment for each action are as follows:

* Reward for BUY action = Transaction cost of the stock bought
* Reward for HOLD action = Holding cost of the current position
* Reward for SELL action = Profit or Loss from selling the open position.

In order to generalize the agent behaviour for different trading situations, the environment regularizes the rewards by clipping at +/- 1. This ensures that the environment does not excessively reward/penalize an action during episodic periods of brief but excessive market volatility.

In addition, the agent (not the environment) also incorporates a discounting function (gamma = 0.97) to recognize the diminishing value of future rewards and adjust for this in the learning model.

* 1. user interface

One way for users to try out the trained model in actual trading scenarios is to use the wrapper python script (Refer to Appendix xx for user guide and demonstration). The script starts by asking user to input historical data for a selected stock. Once completed, the program will run the model and apply actions to the historical data and simulates the profits made based on the environment.

After which, user is prompted to input current date and stock information. Based on the new information, the action value will be generated for each step and the action with the highest action value is recommended to the user.

The program will also advance the model by applying the action to calculate the reward and profit of the next state.

The current state of the program only runs based on a model which only requires user input of the closing price. Once there are other models generated using other technical analysis, the program is easily scalable to include user inputs for that information.

1. model evaluation
   1. features used for evaluation
   2. phase 1
   3. phase 2
   4. phase 3
2. conclusion

**Appendix:**