Single Stock Trading using Reinforcement Learning Algorithms

Predicting Buying, Selling and Holding of Stocks to maximize the profit using Reinforcement Learning techniques

SAN JOSÉ STATE UNIVERSITY

FINAL PROJECT REPORT

Class	CMPE 297 Sec 47
Course	Reinforcement Learning
Teacher	Dr. Jahan Ghofraniha
Project Team Members	• Gulnara Timokhina • Mirsaeid Abolghasemi • Poornapragna Vadiraj • Varun Bhaseen
Date	12/4/2020

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY

1.1 BRIEF PROJECT DESCRIPTION

The primary outcome expected from the project titled "Single Stock Trading using RL algorithms" is a set of comparison to analyze which RL algorithm is better in predicting the action (*Buy, Sell or Hold*) by maximizing the total annual return for a given stock.

The algorithms that were used are suggested as below:

- Q-Actor-Critic (QAC)
- Advantageous-Actor-Critic (A2C)
- Deep-Q-Networks (DQN)
- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)

To ensure we have fairness in comparison, the hyperparameters were same across all the Algorithms except DQN as it requires a *discreet action space* with *finite* values to compute whereas the other algorithms works well in a *box action space* with infinite values.

We implemented two algorithms from scratch *(DQN and Q-AC)* specific to stock trading and have been shared as a separate Jupyter Notebook in this directory. The bottom-up approach was taken to understand the performance of RL algorithm in a trading environment before we could start with an end-to-end project. The other implementations use Stable Baselines project's optimized implementation.

1.2 PROJECT RESULTS

The metrics used to evaluate the algorithms are listed below. The output is received from library pyfolio. The training data is of 10 Years of stock performance of Apple (AAPL). Following are the results:

Algorithm	Time to converge (Minutes)	Sharpe Ratio Metrics	Annual return	Stability
A2C	2.44	2.29	78.81	0.915
PPO	2.42	1.88	101.10	0.908
TRPO	2.13	1.24	50.14	0.888
DQN	120	Not Applicable	10*	Not Applicable
QAC	30	Not Applicable	36.18**	Not Applicable

- *Note: DQN was run separately and had entirely different parameters and environment

- **Note: QAC was also run separately. The hyperparameters were same as the final project colab but metrics from pyfolio (Sharpe ratio and Stability) was not taken under consideration. Also, env was different and not the same trading environment

1.3 CONCLUSION

Based on metrics annual returns and time to converge the best performing model is PPO. Whereas based on Sharpe ratio and stability A2C is better performing.

2 INTRODUCTION

For automatic stock trading, Deep reinforcement learning (DRL) has been accepted as a useful solution. The DRL system is effective to address complex policy challenges by encounters with an uncertain world and thus has two main benefits:

- scalability of portfolio
- independence of business models.

In mathematical finance world, stock market choices are fundamentally complex, namely deciding where to exchange, at what price, and what amount, over a stock market that is extremely stochastic and complex. DRL trading agents construct a multi-factor model and deliver algorithmic trading strategies, which are challenging for human traders, taking into consideration several dynamic financial considerations.

Deep neural networks have found to contribute further. Industry researchers have discussed DRL-fueled trading techniques, as deep neural networks can estimate a return predicted in a state with a certain behavior.

The development of more stable models and techniques would help to efficiently establish general machine learning techniques and DRL methods. DRL was applied for example on nostalgic analyzes on fund distribution and the evaluation of liquidation methods, which demonstrate DRL's ability for different financial tasks. But it is hardly as straightforward to execute a trading plan powered by DRL or RL.

The methods of development and debugging are difficult and error prone. The administration of intermediate trade states, the arrangement of data relevant to training, and standardization of outcomes for measurement measurements are steps of preparation. The implementation steps, for beginners, are time-consuming.

We thus have used a library that is simple to start and has fine-tuned standardized DRL algorithms. It was built according to three main elements:

- **Completeness**: The DRL structure which is a basic prerequisite is entirely protected by our library
- **Hands-on tutorials**: We're looking for a library that's fun for beginners. Detailed tutorials will help users to learn about our library's functionalities
- **Reproducibility**: To ensure accountability and also to give users trust in what they have done, our library promises reproducibility.

The agent participates in a properly specified incentive function in the state space and action space. The entire project is surrounded around Single Stock trading solution.

3 PROBLEM STATEMENT

3.1 BACKGROUND

The future prediction of stock price for gaining more benefits was a challenge in the stock market from the beginning of the financial market establishment. There is a lot of research to find the best way to predict the stock market price trend.

One of the ways is using reinforcement learning. In this way, we have some observation of the stock market, and we want to decide when is the best time to buy, hold or sell stocks. If buying stocks happens before the price rises, the profit (reward) will be positive; otherwise, it will be a negative reward.

The goal of this reinforcement learning approach is to get as much profit as possible. The profit is defined as an *annual return* for the given stock.

3.2 PROBLEMS IN EXISTING TECHNIQUES

In general, there are two main methods to analyze and predict the stock market price. These methods are technical examinations and fundamental evaluations.

- The technical examinations reflect just the historical data of the market to predict the future.
    - These are regression-based evaluations and are not always very accurate as market value is dependent on many features.
- However, the fundamental evaluations look at other data such as news, financial reports, economic status, etc. to determine where the stock prices will fall on any given day.
    - These factors affecting the stock prices can be termed as latent (underlying) features. These are not very well understood and difficult to put into analysis

3.3 REINFORCEMENT LEARNING INTERVENTION

Reinforcement Learning can address the problems stated above by combining both the factors stated above. A policy based RL algorithm can learn from fundamental evaluations (Like PPO, TRPO) whereas a buffer (or memory) based Off-policy RL algorithm (Like DQN) can predict based on technical examinations.

We here are trying to compare how these various techniques are performing on different evaluation criteria. The further work can be done in finetuning of the agents and in preparing an ensemble method which can compensate for both the evaluations required for stock price prediction.

4 PURPOSE

4.1 GOAL

The Primary Purpose of this project is to develop a set of models which can take decision over the trading of a single Stock on whether to Buy , Sell or Hold the stock by maximizing the assets or profits (annual returns).

4.2 SCOPE

Following is the scope of the Project:

- Environment: The environment is a trading environment created on top of default gym environment class. From the library stable baselines, dummy vector environment (DummyVecEnv) is used to create a vertical stack of 3 different environments which are simulated by inheriting Gym Environment class.
- Actions: The actions which are provided in the environment is buying, selling, or holding of stock (no-action)
- Observations: The observations received from the environment are in form of dataset features (close, high, low, open, volume) along with technical indicators like moving average convergence divergence (macd), relative strength index (rsi), Commodity Channel Index (cci), Directional Moving Index (dx)

5 DIFFERENTIATOR

5.1 FROM SCRATCH RL ALGORITHM FOR STOCK MARKET

Initially we created two *from-scratch colab notebooks* for algorithms Q-AC (Q Actor Critic) and DQN (Deep Q Network). This was done to understand and analyze how different RL algorithms can be used for trading stocks.

Due to the time complexity and duration required to run the algorithms we switched to predefined libraries and models provided by the library *stable baselines*. This was helpful as the library provided very optimized prebuilt RL agents with only the parameters which can be tuned.

The colab can be seen here for built from scratch models

5.2 CUSTOMIZING THE STABLE-BASELINE BASED COLAB

The primary colab that was present with the *stable baselines* was customized for Single stock trading. The environment is described in detail in methodology. Following are the key changes that we had done to customize the existing colab:

- Customized the various parameters present for trading environment especially the values for
    - *action space,*
    - *Threshold limit of terminal,*
    - *Stock trading dimensions (increased from 3 to 5),*
    - *reward scaling value (changed from 0.007 to 0.0001) and*
    - *removed additional training and testing environment and focused on only single stock trading environment*

- Customized the existing RL agents and its hyperparameters. The original code had models only for TD3, DDPG, SAC. We removed the existing RL agents and added the following agents:
    - A2C: Advantageous actor critic which was imported from stable baselines library
    - PPO: Proximal policy optimization which was imported from stable baselines ppo2 library
    - TRPO: Trust region policy optimization was also imported from stable baselines library

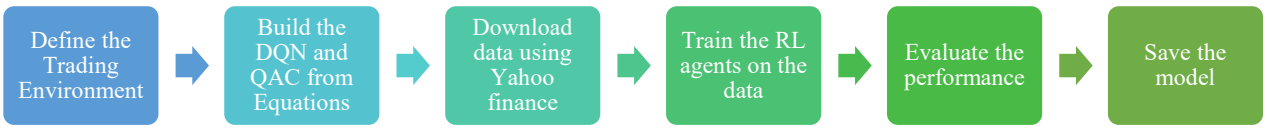
- Adding back testing and model evaluation using pyfolio or TRPO and A2C. This was originally present only for PPO

6 METHODOLOGY

6.1 FROM SCRATCH RL ALGORITHMS

Our Initial Method was to build all the models from scratch and build an ensemble model out of it. But due to time taken to run each episode and then fine tuning of models and parameters we decided to use prebuilt models.

Although we have still built the models Q-AC and DQN from scratch we are not using these models as final agents rather we are using DQN and Q-AC built from scratch as baselines. Below is the approach taken for models built from scratch:



6.1.1 Trading Environment

The environment is defined in a class called *Agent.* The environment is not a simulated environment for entire timeseries data based on gym environment class but rather a simple data frame which is using a combination of observation values, user defined action space, steps, and reward values.

6.1.1.1 Replay Buffer

The environment has a *replay buffer* which can store the historical data and actions taken by the RL agent. The RL agent we are using is DQN and Q Actor critic which are off-policy and combination of off-policy on-policy models. The RL agents hence learns from the historical data. This historical data is seen through the replay buffer.

6.1.1.2 Step

In the *step* function we are adding the state, action, reward, and next state in the memory buffer. The step will continue until and unless the length of the memory is greater than size of batch. At each step, the memory buffer will keep on updating itself with the experiences (reward, action, next state) gathered in each step. The step ends once the entire batch of data has been processed

6.1.1.3 Learn

In the *learn* method the environment takes in experiences from the *step* method and calculates the *loss function.* This is then backpropagated to the neural network (DQN and Critic of Q-AC) and adjust the weights and biases to better predict in the next step based on learning (deviation from actual profit) from the loss function. The backpropagation of loss is carried out using the method *soft_update.*

6.1.1.4 Observation

The observation is returned from *window* method:

- N past bars(windows), where each has open, high, low, and close prices
- An indication that the share was bought some time ago
- Profit or loss that we currently have from our current position

6.1.2 Building the Model

The models that we are using for predicting the stock values are Deep Q Network (DQN) and Q Actor critic (QAC).

6.1.2.1 Q-AC

Both the Critic and Actor functions are parameterized with neural networks. Since the Critic neural network parameterizes the Q value, it is called Q Actor Critic.

- ***Actor***: The Actor is the policy-based model: which takes the Stock Price as an input and gives out a log probability distribution of actions: Hold, Buy, or Sell.
- ***Critic***: The Critic is a DQN which takes the Stock Price as input and an Action given by the Actor and computes the Q value.
- ***Q-Value***: The Critic is trained to maximize the Q Value. The Q value is used by the Actor to improve probabilities of better actions.

Model can be defined in simple terms from below mentioned equation. The final equation for Q-AC can be seen in the red block:

- The Score function in the REINFORCE is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t]$$

which can be written as :

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_t,a_t}[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)] \mathbb{E}_{r_{t+1},s_{t+1},\ldots,r_T,s_T}[G_t]$$

- But the Expectation term is the Q-Value:

$$\mathbb{E}_{r_{t+1},s_{t+1},\ldots,r_T,s_T}[G_t] = Q(s_t, a_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_t,a_t}[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)] Q_w(s_t, a_t)$$

$$= \mathbb{E}_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) Q_w(s_t, a_t)]$$

Plugging the Q-Value in the main Equation

6.1.2.2 DQN

Q-learning is a value-based Reinforcement Learning algorithm that is used to find the optimal action-selection policy using a Q function. In deep Q-learning, we use a neural network to approximate the Q-value function. The *state* is given as the input and the Q-value of allowed *actions* (Buy, Sell, Hold) is the predicted output.

DQN is an extension of Q-learning algorithm that uses a neural network to represent the Q value. It leverages a Neural Network to estimate the Q-value function. Like supervised (deep) learning, in DQN we train a neural network and try to minimize a *loss function*. A DQN agent choose an

action according to the greedy policy - maximizing the Q* function (in our case maximizing the profit). The below equation defines the overall DQN agent.

$$Q^*(s_t, a) \rightarrow r(s_t, a_t) + \gamma max_a Q^*(s_{t+1}, a) \qquad (1)$$

$Q^* = Optimal\ Q\ Value$

$\gamma = discount\ factor$

$s = state, a = action, r = reward, t = time$

6.1.3 Yahoo Finance Data Download

The observation values for the environment comes from the Yahoo finance library called *yfinance.* This library helps us to download the data directly by consuming yahoo API based on the specified date range. The data is stored as a csv file which is consumed by the environment at runtime.

The key features that we get for the data is open, high, low, and close for the specified stock (in our case Apple stock 'appl')

6.1.4 Training Loop

The training loop to train the model for DQN and Q AC is dependent on number of episodes. The return value that we get is the total profit achieved after n_steps for the entire dataset in 2 episodes. It needs to be noted that due to time taken to converge which was close to 120 minutes for DQN and 30 Minutes for Q-AC we could not exceed the number of episodes.

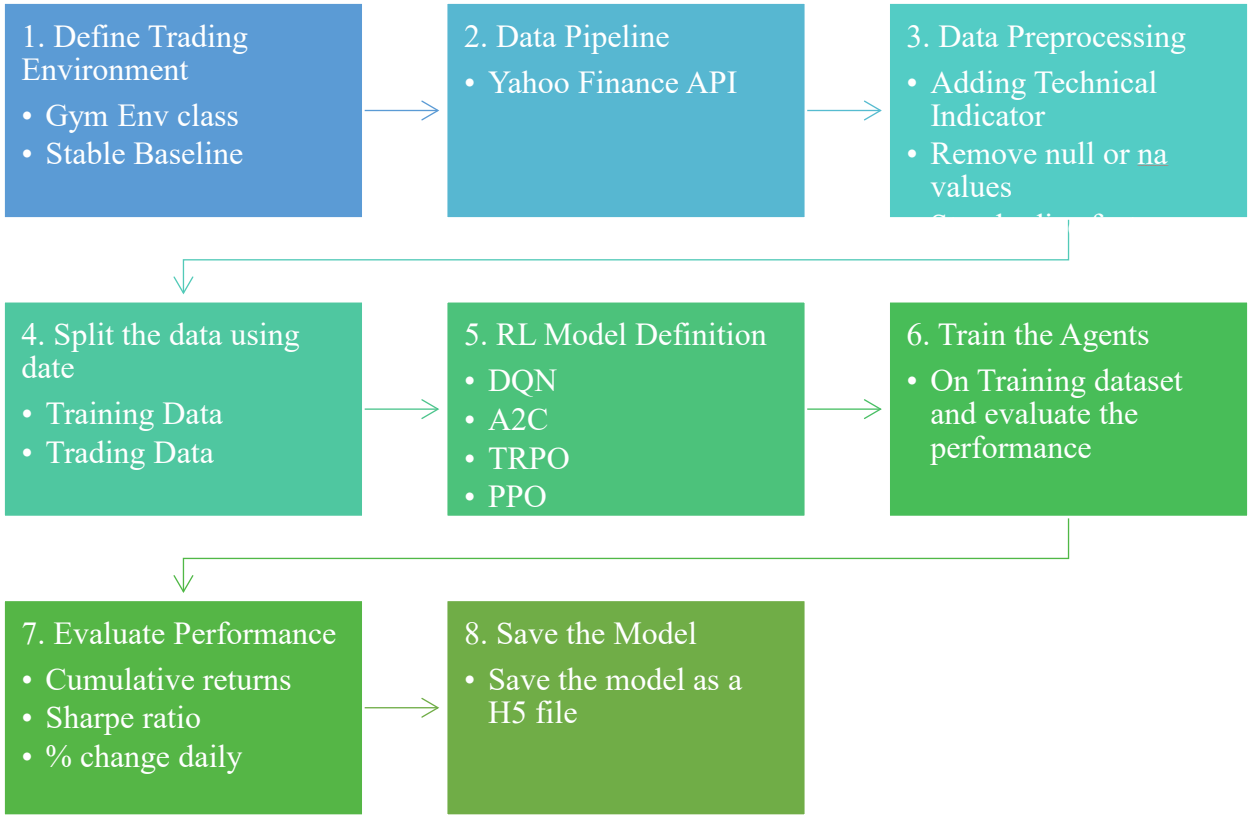
6.1.5 Performance Evaluation

The performance evaluation for models was done on profit achieved in each session and cumulative profit achieved in the entire episode. The performance is evaluated using Tensor Board (*class: TensorBoardLogger*)

- ***Profit***: The profit seen in code is the profit achieved at each step after closing of the stocks in the day and trading done.
- ***Total Profit***: This is the cumulative profit that is achieved in the entire episode for the entire dataset. We ignore this value since this does not reflect the model performance in day-to-day trading sessions.

Performance brief can be seen in section conclusion. Whereas the detailed results and tensor board outputs can be seen from section implementation results

6.2 SINGLE STOCK TRADING METHODOLOGY



6.2.1 Define Trading Environment

6.2.1.1 Approach

Considering the stochastic and interactive nature of the automated stock trading tasks, a financial task is modeled as a Markov Decision Process (MDP) problem. The training process involves observing stock price change, taking an action and reward's calculation to have the agent adjusting its strategy accordingly. By interacting with the environment, the trading agent will derive a trading strategy with the maximized rewards as time proceeds.

Our trading environments, based on OpenAI Gym framework, simulate live, stock markets with real market data according to the principle of time-driven simulation.

6.2.1.2 Action Space

The action space describes the allowed actions that the agent interacts with the environment. Normally, action a includes three actions: {-1, 0, 1}, where -1, 0, 1 represent selling, holding, and buying one share.

We use an action space {-k, -1, 0, 1, k}, where k denotes the number of shares to buy and -k denotes the number of shares to sell. For example, "Buy 10 shares of AAPL" or "Sell 10 shares of AAPL" are 10 or -10, respectively. The continuous action space needs to be normalized to [-1, 1],

since the policy is defined on a Gaussian distribution, which needs to be normalized and symmetric.

6.2.1.3 Observation Space

The Observation space: Observations are pieces of information that environment gives the agent on every timestamp besides the reward. Here the environment is provided the agent a set of numbers from 0 to NumPy infinity and shape is from state space. The state space is defined in dummy class and is an integer of shape defined by stock dimension which in our case is a tensor of shape 8.

The observation space returns the set of observations. The observation will be the current price of a stock, various stock features (low, high, open, close, volume), and technical Indicator list (macd, cci, rsi, dx) values.

6.2.1.4 Step

Each step will return a reward, state, and terminal. Steps are important because it tells the program to terminate of agent does not converge after fixed steps. Currently we have a max step of 100 set as default. This can be changed by changing hmax value in dummy env class stated above during initialization

6.2.1.5 Reward and Reward Scaling

Reward scaling defines the scaling factor of reward. In the environment the scaling factor is defaulted to 0.0001

Rewards from the environment can be defined as sum of state value and sum of trading done in the given step (profit or loss achieved from buying or selling of shares) minus the total amount during the start of the step.

To maximize the reward, we want to have a positive value at the end of the step. For example:

- The agent is provided with $100K then, the agent takes this amount and starts trading.
- At each trading instance the agent gets the observation in form open, close, high, low. Volume, macd, cci, rsi, dx
- Based on observation the agent carries out the trading in form of selling the stocks or buying the stocks.
- The reward is defaulted at every 100-steps .
- It means after 100 steps the reward will be the Money left after subtraction of initial value which in this example is value obtained after subtracting $100K.
- The objective is to maximize this reward and hence the total profit or in proper finance terms called as annual return

6.2.2 Create Data Pipeline

A Yahoo Data Pipeline was developed using the library "*yfinance"*. The trading environment needs a set of data that it can use to simulate the environment. The trading data can be downloaded directly from using the yahoo finance based on ticker, start-date, and end-date. The ticker defines the stock name or index that we want to download the data from which in our case is Apple stock.

The class name defined is called *YahooDownloader*, to collect data from Yahoo API. We have two methods defined in this class as stated below:

- Fetch Data: This is used to fetch data from the API and returns a Pandas data frame with seven columns for the specified stock ticker:
    - date
    - open
    - high
    - low
    - close
    - volume
    - tick symbol
- Select Equal Rows Stock: This method takes in the data frame and resets the index. This is done after feature engineering mostly to drop any *na* or *null* values. The method returns a data frame with equal number of rows for given ticker (stock name or index).

6.2.3 Data Preprocessing

The data that we get from *YahooDownloader* class is structured but not preprocessed, the dataset may still have some *null* or *na* values. Also, it lacks some technical indicators which our RL agents can rely on to take better decision on managing the stock. We have created a *FeatureEngineer* class to cleanse and preprocess the data and add various features as described below.

This class has features for preprocessing the data for stock price that we have downloaded from Yahoo Finance library. The preprocessing is required so that we can standardize the data from different stock indexes (like S&P, DJI or Nasdaq) and remove any *na* or *null* values from the dataset. We can remove and drop such rows and have a uniform data.

The key methods of this class are:

- ***preprocess_data***: Fills the missing values at beginning and at end. Checks if any additional indicators are missing and needs to be added from *stockstats* library. Also checks if turbulence is True or not and If true then add turbulence value.
- ***_add_technical_indicator***: Add the following technical indicators from *stockstats* library ['macd', 'rsi_30', 'cci_30', 'dx_30'] for the given stock. These stand as 'moving average convergence divergence', 'relative strength index', 'Commodity Channel Index', 'Directional Moving Index'. These values are very important for RL Agent to learn and understand the actions that needs to be taken on stock based on strength indicators.
- ***_add_turbulence***: Calculate turbulence from the *_calculate_turbulence* method. Then add the turbulence for the respective stock. Turbulence in the stock market means volatility. Higher Volatility means frightened investor and higher selling for the stock. Whereas lower volatility means calm investor and holding of stock.
- ***_Calculate_turbulence***: Here the turbulence index is calculated based on VIX from Dow Jones Industrial Average. This value will help to decide the volatility of a particular stock in market and enable RL with additional information of Buying or selling of stock.

- ***get_type_list***: This method checks what type of features are available in the data set. That is downloaded from Yahoo Finance.
    - If number of Features is 1 then they are only the ["close"] value of stock
    - Feature = 2: ["close", "high"]
    - Feature = 3: ["close", "high", "low"]
    - Feature = 4: ["close", "high", "low", "open"]
    - Feature = 5: ["close", "high", "low", "open", "volume"]

6.2.4 Splitting the Dataset

We have method *data_split* which will help us to split the entire preprocessed data into Training and Testing data. The criteria used for splitting of dataset is date. Splitting of data is useful as it helps to check the performance of RL agents on a live dataset and tune the performance to check for overfitting or underfitting of predictions before they can be moved to production.

6.2.5 Model Definition

In this stage the RL agents are defined that will be run on the Stock Trading environment. The models are not created from scratch and rather used as a **prebuilt model** from ***stable-baselines*** library. The model classes are imported from the library and only the parameters need to be defined like Number of steps, environment, gamma (discount actor), entropy coefficients, learning rate, alpha, epsilon etc.

Following are the models that has been taken into consideration:

- A2C: Advantage Actor Critic
- TRPO: Trust Region Policy Optimization
- PPO: Proximal Policy Optimization

Note: DQN and Q-AC models which are built from scratch are explained in next section 6.2.

6.2.6 Performance Evaluation

The models are evaluated based on financial metrics provided by the library pyfolio. The metrics that we are focusing on most is:

- Daily return: defined in method *backtest_strat*
- Percent change in closing from previous day: defined in method *baseline_strat*
- Annual return and Sharpe ratio: defined in method *get_daily_return*

The stock index used for baselining the value is DJI (Dow Jones Industrial average). Although it can be replaced with any index albeit S&P, Nasdaq and so on.

The overall performance of the models can be seen in section results. Whereas the performance brief can be seen in section conclusion

7 ARCHITECTURE DIAGRAMS

7.1 CLASS DIAGRAMS

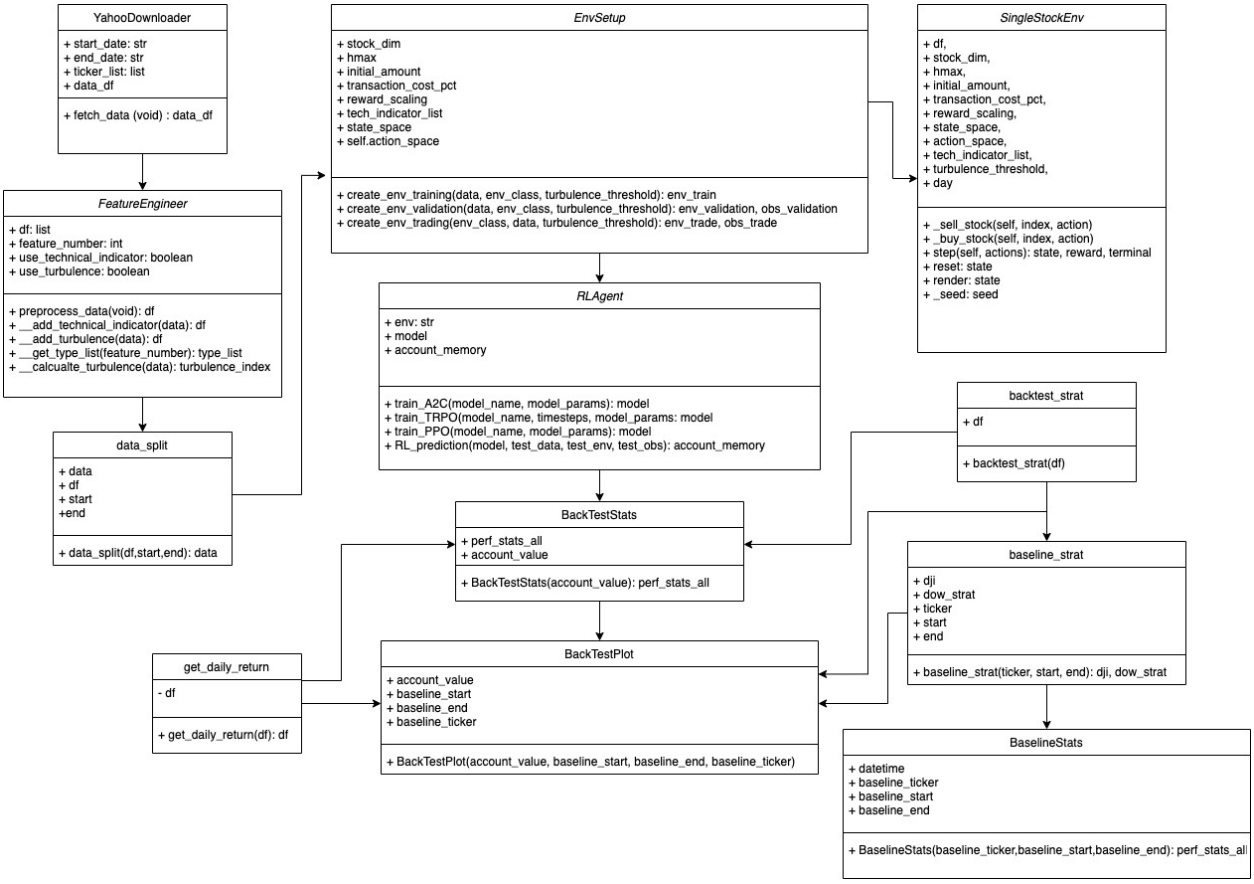


Figure 1: Class Diagram for Single Stock trading Project using RL colab

7.2 RL AGENT ARCHITECTURE DIAGRAMS

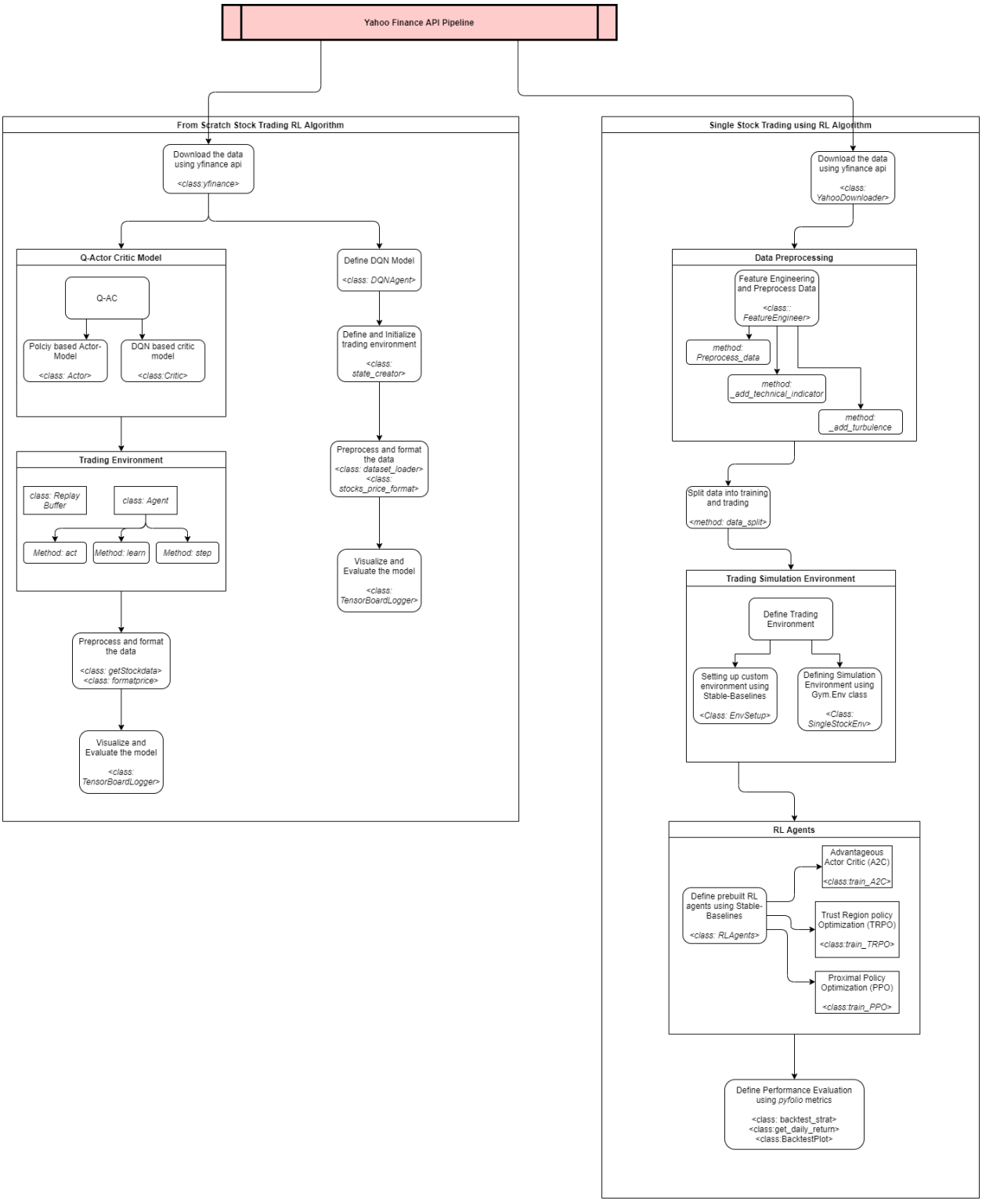


Figure 2: RL Agents Architecture Diagram for both Models

8 PROJECT AND RESOURCE MANAGEMENT

8.1 PROJECT MANAGEMENT

The entire project management was carried out using Trello Board. Below is the screenshot for the Trello board. The entire task was managed using Kanban approach.

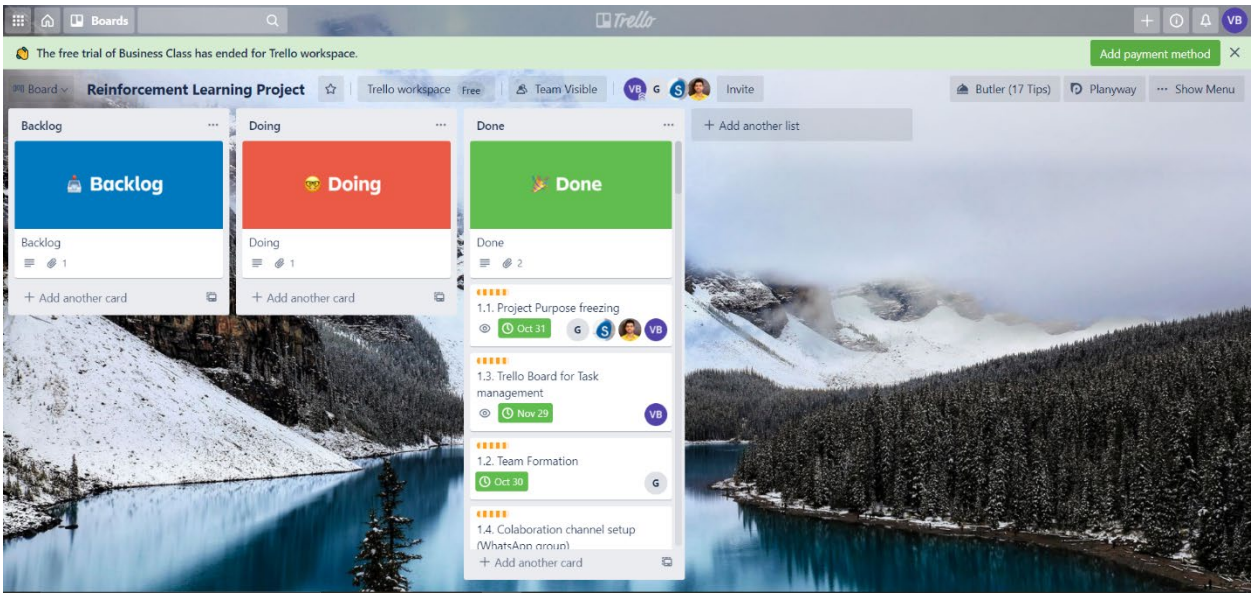


Figure 3 Snapshot for Trello Board

8.2 PROJECT PLAN

Milestone Description	Assigned To	Start	Status
1. Handshake and Collaboration Setup		10/24/2020	
1.1. Project Purpose freezing	Team	10/30/2020	Done
1.2. Team Formation	Gulnara Timokhina	10/24/2020	Done
1.3. Trello Board for Task management	Varun Bhaseen	10/27/2020	Done
1.4. Colaboration channel setup (WhatsApp group)	Poornapragna Vadiraj	10/27/2020	Done
1.5. MS One Drive and Google Drive setup	Mirsaeid Abolghasemi	11/2/2020	Done
2. Project Kick-off		10/27/2020	Done
2.1. Literature Research	Mirsaeid Abolghasemi	11/5/2020	Done
2.2. Problem Identification	Gulnara Timokhina	11/1/2020	Done
2.3. Project Scoping	Poornapragna Vadiraj	11/2/2020	Done
2.4. Abstract Draft	Varun Bhaseen	10/27/2020	Done
2.5. Abstract Submission	Team	10/30/2020	Done
3. DQN Baseline for Stock Prediction		11/5/2020	
3.1. Setup the Development and Testing Environment Locally for DQN	Mirsaeid Abolghasemi	11/7/2020	Done
3.1.1. Setup Anaconda environment with all required libraries for Deep Learning (PyTorch, Tensorboard)	Mirsaeid Abolghasemi	11/5/2020	Done
3.1.2. Install the Open AI Gym and all dependencies	Mirsaeid Abolghasemi	11/7/2020	Done
3.2. Identify the Target Label in the Open AI Gym environment (StockTradingENV: GOOGL) for DQN	Mirsaeid Abolghasemi	11/12/2020	Done
3.3. Setup and Train agent using DQN on the Environment	Mirsaeid Abolghasemi	11/13/2020	Done

Milestone Description	Assigned To	Start	Status
3.4. Hyperparameter Tuning for different parameters of DQN	Mirsaeid Abolghasemi	11/14/2020	Done
3.5. Record DQN results in Tensorboardx and publish link to teammates with logs	Mirsaeid Abolghasemi	11/13/2020	Done
4. Using Actor-Critic for Stock Prediction		11/7/2020	
4.1. Setup the Development and Testing Environment Locally fro Actor Critic	Gulnara Timokhina	11/7/2020	Done
4.1.1. Setup Anaconda environment with all required libraries for Deep Learning (PyTorch, Tensorboard)	Gulnara Timokhina	11/15/2020	Done
4.1.2. Install the Open AI Gym and all dependencies	Gulnara Timokhina	11/13/2020	Done
4.2. Identify the Target Label in the environment (StockTradingENV: GOOGL) for Actor Critic	Gulnara Timokhina	11/15/2020	Done
4.3. Setup and Train agent on the Environment using Actor-Critic	Gulnara Timokhina	11/20/2020	Done
4.4. Hyperparameter Tuning for different parameters of Actor Critic	Gulnara Timokhina	11/21/2020	Done
4.5. Record results of Actor Critic in Tensorboardx and publish link to teammates with logs	Gulnara Timokhina	11/22/2020	Done
5. Using TRPO (Trust Region Policy Optimization) for Stock Prediction		11/21/2020	
5.1. Setup the Development and Testing Environment Locally for TRPO	Poornapragna Vadiraj	11/7/2020	Done
5.1.1. Setup Anaconda environment with all required libraries for Deep Learning (PyTorch, Tensorboard)	Poornapragna Vadiraj	11/15/2020	Done
5.1.2. Install the Open AI Gym and all dependencies	Poornapragna Vadiraj	11/13/2020	Done
5.2. Identify the Target Label in the environment (StockTradingENV: GOOGL) for TRPO	Poornapragna Vadiraj	11/15/2020	Done
5.3. Setup and Train agent on the Environment using TRPO	Poornapragna Vadiraj	11/20/2020	Done
5.4. Hyperparameter Tuning for different parameters of TRPO	Poornapragna Vadiraj	11/21/2020	Done
5.5. Record results of TRPO in Tensorboardx and publish link to teammates with logs	Poornapragna Vadiraj	11/22/2020	Done
6. Using PPO (Proximal Policy Optimization) for Stock Prediction			
6.1. Setup the Development and Testing Environment Locally for PPO	Varun Bhaseen	11/7/2020	Done
6.1.1. Setup Anaconda environment with all required libraries for Deep Learning (PyTorch, Tensorboard)	Varun Bhaseen	11/15/2020	Done
6.1.2. Install the Open AI Gym and all dependencies	Varun Bhaseen	11/13/2020	Done
6.2. Identify the Target Label in the environment (StockTradingENV: GOOGL) for PPO	Varun Bhaseen	11/15/2020	Done
6.3. Setup and Train agent on the Environment using PPO	Varun Bhaseen	11/20/2020	Done
6.4. Hyperparameter Tuning for different parameters of PPO	Varun Bhaseen	11/21/2020	Done
6.5. Record results of PPO in Tensorboardx and publish link to teammates with logs	Varun Bhaseen	11/22/2020	Done
7. Performance Comparison of Agents			
7.1. Integrating all approaches in single colab	Varun Bhaseen	11/27/2020	Done
7.2. Compare the accuracy of each technique for the trained agents	Varun Bhaseen	11/27/2020	Done
7.3. Time to convergence to most optimal prediction by each agent for each technique	Varun Bhaseen	12/2/2020	Done

Milestone Description	Assigned To	Start	Status
7.4. Setting Hyper tuning parameters for all agents (TRPO, PPO, A2C)	Gulnara Timokhina	12/2/2020	Done
7.5. Develop and publish interactive visualization for each result and draw conclusions	Poornapragna Vadiraj	12/2/2020	Done
7.6. Evaluate results and plot comparisons for DQN	Mirsaeid Abolghasemi	12/2/2020	Done
8. Project presentation and closure	Team	12/2/2020	Done

8.3 PROJECT TIMELINE

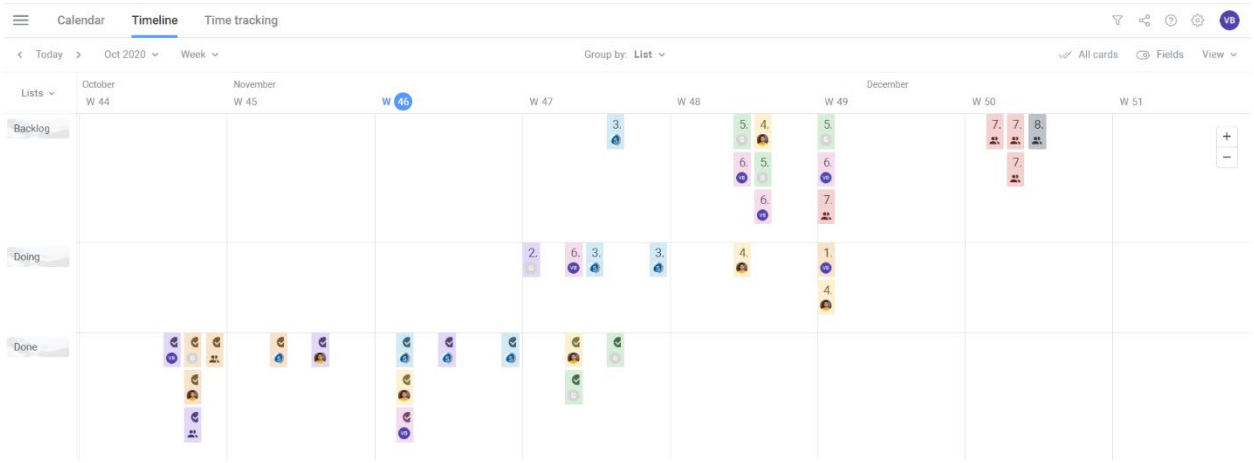


Figure 4: Trello Project Timeline Report (Source: Trello board for Reinforcement Learning Project)

8.4 PROJECT CONTRIBUTION AND OWNERSHIP

Project Implementation Plan	Grand Total
Gulnara Timokhina	**38**
1.2. Team Formation	6
2.2. Problem Identification	4
4.1. Setup the Development and Testing Environment Locally fro Actor Critic	2
4.1.1. Setup Anaconda environment with all required libraries for Deep Learning (PyTorch, Tensorboard)	3
4.1.2. Install the Open AI Gym and all dependencies	2
4.2. Identify the Target Label in the environment (StockTradingENV: Appl) for Actor Critic	2
4.3. Setup and Train agent on the Environment using Actor-Critic	5
4.4. Hyperparameter Tuning for different parameters of Actor Critic	5
4.5. Record results of Actor Critic in Tensorboardx and publish link to teammates with logs	7
7.4. Develop and publish interactive visualization for each result and draw conclusions	2
Mirsaeid Abolghasemi	**35**
1.5. MS One Drive and Google Drive setup	1
2.1. Literature Research	6
3.1. Setup the Development and Testing Environment Locally for DQN	2
3.1.1. Setup Anaconda environment with all required libraries for Deep Learning (PyTorch, Tensorboard)	3
3.1.2. Install the Open AI Gym and all dependencies	2
3.2. Identify the Target Label in the Open AI Gym environment (StockTradingENV: Appl) for DQN	2
3.3. Setup and Train agent using DQN on the Environment	5
3.4. Hyperparameter Tuning for different parameters of DQN	5

Project Implementation Plan	Grand Total
3.5. Record DQN results in Tensorboardx and publish link to teammates with logs	7
7.6. Evaluate results and plot comparisons for DQN	2
Poornapragna Vadiraj	**34**
1.4. Colaboration channel setup (WhatsApp group)	3
2.3. Project Scoping	3
5.1. Setup the Development and Testing Environment Locally for TRPO	2
5.1.1. Setup Anaconda environment with all required libraries for Deep Learning (PyTorch, Tensorboard)	3
5.1.2. Install the Open AI Gym and all dependencies	2
5.2. Identify the Target Label in the environment (StockTradingENV: Appl) for TRPO	2
5.3. Setup and Train agent on the Environment using TRPO	5
5.4. Hyperparameter Tuning for different parameters of TRPO	5
5.5. Record results of TRPO in Pyfolio and publish link to teammates with logs	7
7.5. Develop and publish interactive visualization for each result and draw conclusions	2
Team	**5**
1.1. Project Purpose freezing	2
2.5. Abstract Submission	1
8. Project presentation and closure	2
Varun Bhaseen	**45**
1.3. Trello Board for Task management	2
2.4. Abstract Draft	4
6.1. Setup the Development and Testing Environment Locally for PPO	2
6.1.1. Setup Anaconda environment with all required libraries for Deep Learning (PyTorch, Tensorboard)	3
6.1.2. Install the Open AI Gym and all dependencies	2
6.2. Identify the Target Label in the environment (StockTradingENV: Appl) for PPO	2
6.3. Setup and Train agent on the Environment using PPO	5
6.4. Hyperparameter Tuning for different parameters of PPO	5
6.5. Record results of PPO in Pyfolio and publish link to teammates with logs	7
7.1. Compare the accuracy of each technique for the trained agents	2
7.2. Time to convergence to most optimal prediction by each agent for each technique	5
7.3. Performance Evaluation for each Hypertuning instance for all agents	6
Grand Total	**157**

9 RESULTS AND CONCLUSION

9.1 HYPERPARAMETERS USED FOR RL AGENTS

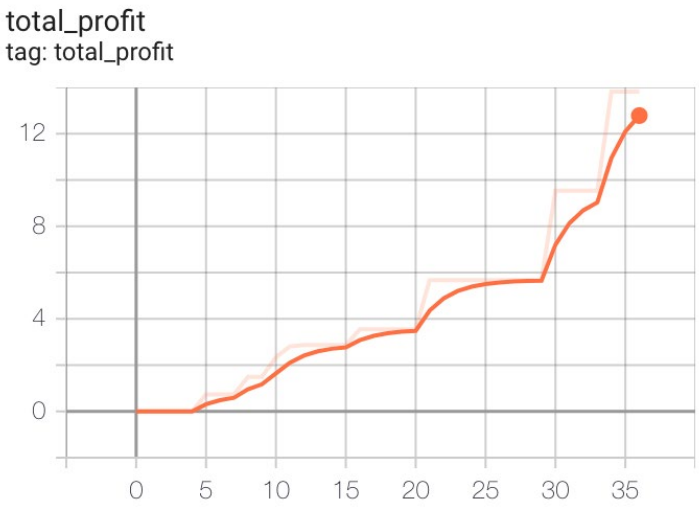
Following are the hyperparameters used for the RL algorithms to ensure that we can have a common platform for comparison of each of their performances.

Hyperparameters Used			
Learning rate = 0.0007	Policy = 'MlpPolicy'	Time steps = 100000	Entropy coefficient = 0.005
Beta (Discount factor) = 0.99	N_steps = 128	Minibatches = 4	

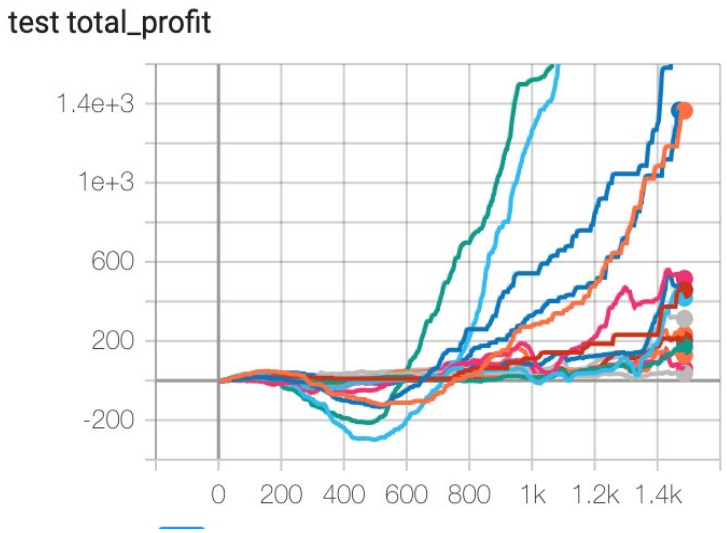
9.2 RESULTS

9.2.1 DQN TensorBoard Performance Chart

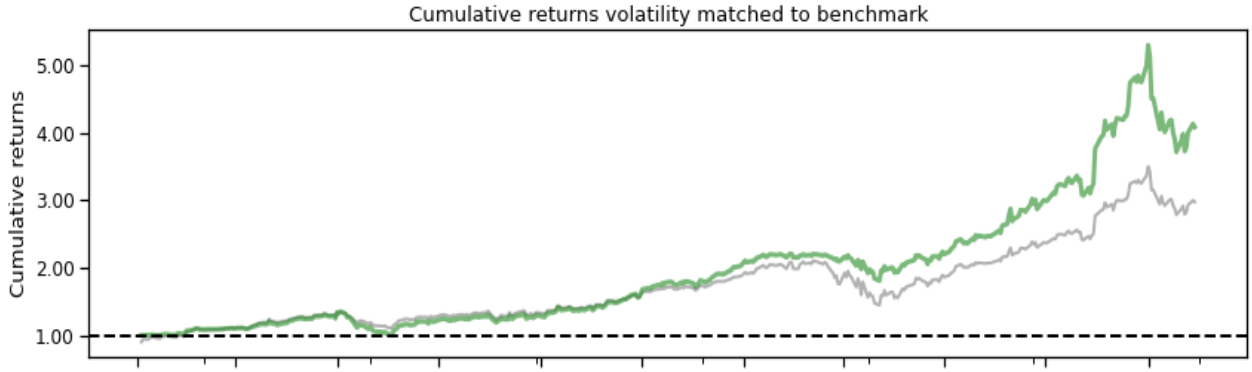
total_profit



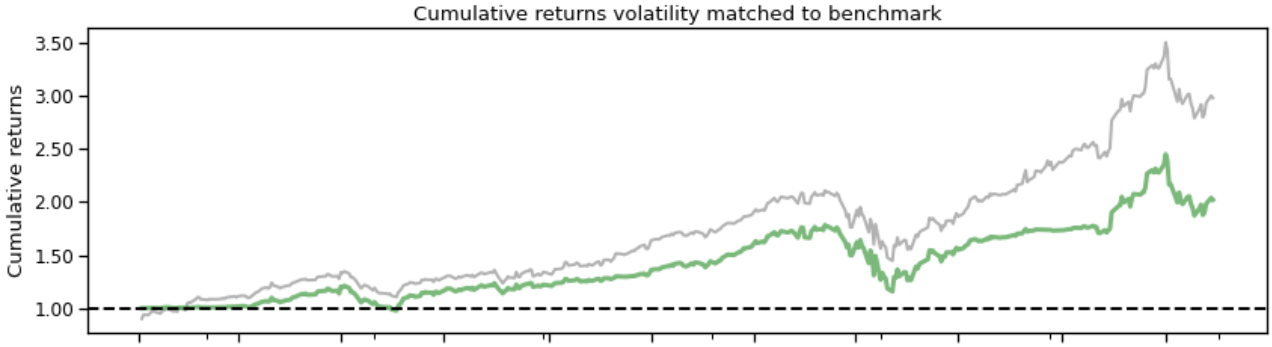
9.2.2 Q-AC Tensorboard Performance Chart



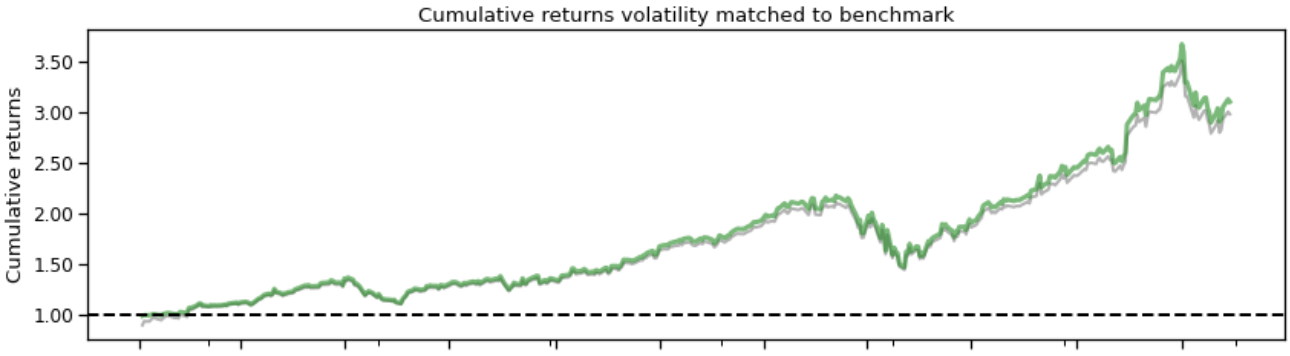
9.2.3 A2C Pyfolio Financial Metrics chart



9.2.4 TRPO Pyfolio Financial Metrics Chart



9.2.5 PPO Pyfolio Financial Metrics Chart



9.3 ANALYSIS

Following is the key analysis for the performance of all the RL Algorithm segregated by the common metrics and based on common parameters:

Algorithm	Time to converge (Minutes)	Sharpe Ratio Metrics	Annual return	Stability
A2C	2.44	2.29	78.81	0.915
PPO	2.42	1.88	101.10	0.908
TRPO	2.13	1.24	50.14	0.888
DQN	120	Not Applicable	10*	Not Applicable
QAC	30	Not Applicable	36.18**	Not Applicable

- *Note: DQN was run separately and had entirely different parameters and environment
- **Note: QAC was also run separately. The hyperparameters were same as the final project colab but metrics from pyfolio (Sharpe ratio and Stability) was not taken under consideration. Also, env was different and not the same trading environment

9.4 CONCLUSION

Based on the Analysis above we can clearly say that if annual returns is the criteria for a User or web application then PPO performed the best. Further results and explanations can be seen below:

- Time to Converge: The best performing model in terms of time to converge was PPO. This was expected as PPO also happens to be simpler and due to its log properties and clipping of regions it was expected to be fastest. This was followed up with A2C and TRPO
- Sharpe Ratio: The best model in terms of sharpe ratio was A2C followed up by PPO and then TRPO. This was occurring because of A2C's critic model sending the most optimal feedback on policy optimization to the actor model. Hence the performance was much higher.

- Annual Returns: The highest annual returns were achieved from PPO and this was followed up by A2C and TRPO. PPO most closely followed the market index and was best returning due to the quicker response and focus area limited to few factors like macd, rsi, and dx
- Stability: In terms of stability of model the best performing agent was A2C and then it was closely followed up PPO and TRPO

10 APPENDIX

10.1 REFERENCES

- Foy, P. (2020, November 27). Deep Reinforcement Learning for Trading with TensorFlow 2.0. Retrieved December 08, 2020, from https://www.mlq.ai/deep-reinforcement-learning-for-trading-with-tensorflow-2-0/
- Eer, K. (2020, September 14). Reinforcement Learning for Stock Trading Agent. Retrieved December 08, 2020, from https://medium.com/@kaijuneer/reinforcement-learning-for-stock-trading-agent-20d89a1a280a
- Kung-Hsiang, H. (2018, September 16). Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG). Retrieved December 08, 2020, from https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287
- Yang, B. (2020). AI4Finance-LLC/FinRL-Library. Retrieved December 08, 2020, from https://github.com/AI4Finance-LLC/FinRL-Library/tree/master/finrl
- Yang, B. (2020). Google Colaboratory. Retrieved December 08, 2020, from https://colab.research.google.com/github/AI4Finance-LLC/FinRL-Library/blob/master/FinRL_single_stock_trading.ipynb
- Avoy D., Yang, B. (2020). AI4Finance-LLC/Deep-Reinforcement-Learning-for-Automated-Stock-Trading-Ensemble-Strategy-ICAIF-2020. Retrieved December 08, 2020, from https://github.com/AI4Finance-LLC/Deep-Reinforcement-Learning-for-Automated-Stock-Trading-Ensemble-Strategy-ICAIF-2020/blob/master/config/config.py
- Yang, B. (2020). AI4Finance-LLC/Deep-Reinforcement-Learning-for-Automated-Stock-Trading-Ensemble-Strategy-ICAIF-2020. Retrieved December 08, 2020, from https://github.com/AI4Finance-LLC/Deep-Reinforcement-Learning-for-Automated-Stock-Trading-Ensemble-Strategy-ICAIF-2020
- Yang, B. (2020, December 02). Deep Reinforcement Learning for Automated Stock Trading. Retrieved December 08, 2020, from https://towardsdatascience.com/deep-reinforcement-learning-for-automated-stock-trading-f1dad0126a02
- Ekta15, A. (2020, November 24). Predicting Stock Prices using Reinforcement Learning (with Python Code!) -. Retrieved December 08, 2020, from https://www.analyticsvidhya.com/blog/2020/10/reinforcement-learning-stock-price-prediction/

10.2 SOURCE CODE REPOSITORY LINKS

GitHub Link for Source Code and all Deliverables:

GPSV-Project/Reinforcement-Learning-Stock-Prediction: CMPE 297 sec 47 Project for Reinforcement Learning (github.com)

~~~End Of Document~~~
~~~