CSE4063 - Reinforcement Learning

Project Report

Trading of Stocks and Cryptocurrencies using Reinforcement Learning

By

19BAI1150	Abhiraj Chaudhary
19BAI1157	Yash Tripathi
19BAI1118	Ashwin U Iyer
19BAI1129	Hitesh Goyal

B. Tech Computer Science and Engineering

Submitted to

Dr. A Nayeemulla Khan

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

April 2022

DECLARATION

We hereby declare that the report titled "**Trading of Stocks and Cryptocurrencies using Reinforcement Learning"** submitted by us to VIT Chennai is a record of bona-fide work undertaken by us under the supervision of **Dr. A Nayeemulla Khan**, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai.

Abhiraj Chaudhary
19BAI1150

Yash Tripathi
19BAI1157

Hitesh Goyal
19BAI129

Ashwin U Iyer
19BAI1118

CERTIFICATE

Certified that this project report entitled "**Trading of Stocks and Cryptocurrencies using Reinforcement Learning"** is a bonafide work of **Abhiraj Chaudhary - 19BAI1150, Yash Tripathi - 19BAI1157, Ashwin U Iyer - 19BAI1118, Hitesh Goyal - 19BAI1129** and they carried out the Project work under my supervision and guidance for CSE4063 - Reinforcement Learning.

Dr. A Nayeemulla Khan
SCOPE, VIT Chennai

ABSTRACT

Traditionally stock market trading has been done using human resources and has been conducted using human intelligence. Naturally human intelligence has certain limitations which would make the whole process quite inefficient. To make profits in stock trading you have to learn and understand about the stock you are about to invest in and then you have to formulate strategies to invest in the market. This whole process if done by a human will be quite slow and will also generate a small amount of profits per hour. Using an Automated solution to this problem would help us alleviate the issues which plague the existing system. Automating the learning process would accelerate the process of creating the strategy. Reinforcement learning helps us to solve this problem by providing a way to learn the strategy and then executing the action in a strategic manner. In this project we will explore the different ways we can perform automatic trading in the NIFTY stock market environment and in the Ethereum USD market and observe how different models perform in the given environment.

CONTENTS

1. Introduction

Trading stocks and crypto-currencies can be a very viable source of income. Most people in the market prefer investing in companies where experts diversify their assets to gain profits. The work of experts in this field is to analyze the trends of the market and see what the best way to go is.

Training an RL agent which is capable of understanding these patterns and performing appropriate actions can easily automate this process and increase its availability. RL is a very relevant solution to this problem simply because it optimizes rewards in an environment. So, if the market is the environment and the profits are the reward, performing RL is a genuine possibility.

1.1 **Objective and goal of the project**

The objective of the project is to make an RL agent which is capable of making trading decisions which end up giving profit on the investments. For doing so, we must explore the various possibilities in terms of agents available, their learning capabilities and their relevance to our use case to see what works best.

1.2 **Problem Statement**

The stock and crypto market are very prominent in the current world. Making good investments in these markets can be a very risky but lucrative venture. To reduce the risk, and taking calculated guesses, using RL to train a model can be a viable solution. Try to use RL techniques to train an agent which can function in a stock and one in a cryptocurrency market while being profitable.

2. Literature Survey

Jiang et al. (2017) translated cryptocurrency into a portfolio management process, which later on used decision-making to simplify all of it. The paper uses a model-less convolutional neural network with historic prices of a set of financial assets as its input, outputting portfolio weights of the set. The network is trained with 0.7 years' price data from a cryptocurrency exchange. The training is done in a reinforcement manner, maximizing the cumulative return, which is regarded as the reward function of the network. Back test trading experiments with a trading period of 30 minutes is conducted in the same market, achieving 10-fold returns in 1.8 month's periods.

Jae Won Lee et al. (2001) paper proposed a method of applying reinforcement learning, suitable for modeling and learning various kinds of interactions in real situations, to the problem of stock price prediction. The stock price prediction problem considered as the Markov process can be optimized by a reinforcement learning based algorithm. TD(0), a reinforcement learning algorithm which learns only from experiences, is adopted and function approximation by an artificial neural network is performed to learn the values of states each of which corresponds to a stock price trend at a given time. An experimental result based on the Korean stock market is presented to evaluate the performance of the proposed method.

Xing Wu et al. (2020) for the time-series nature of stock market data, the Gated Recurrent Unit (GRU) is applied to extract informative financial features, which can represent the intrinsic characteristics of the stock market for adaptive trading decisions. Furthermore, with the tailored design of state and action spaces, two trading strategies with reinforcement learning methods are proposed as GDQN (Gated Deep Q-learning trading strategy) and GDPG (Gated Deterministic Policy Gradient trading strategy). To verify the robustness and effectiveness of GDQN and GDPG, they are tested in trending and volatile stock markets from different countries. Experimental results show that the proposed GDQN and GDPG not only outperform the Turtle trading strategy but also achieve more stable returns than a state-of-the-art direct reinforcement learning method, DRL trading strategy, in the volatile stock market. As far as the GDQN and the GDPG are compared, experimental results demonstrate that the GDPG with an actor-critic framework is more stable than the GDQN with a critic-only framework in the ever-evolving stock market.

We chose a different, more traditional method to increase our understanding of Reinforcement Learning. Allowing us to interpret model free methodology in a dynamic calculative environment.

3 Requirements Specification

3.1 Hardware Requirements

2GB GPU Memory, 500mb of secondary memory, 8gb ram.

3.2 Software Requirements

Python 3, Tensorflow, openAi-gym, StableBaselines3, numpy, pandas.

4 System Design

4.1 Environments

Trading algorithms are mostly implemented in two markets: Cryptocurrency and Stock. AnyTrading aims to provide some Gym environments to improve and facilitate the procedure of developing and testing RL-based algorithms in this area. This purpose is obtained by implementing three Gym environments: TradingEnv, ForexEnv, and StocksEnv.

TradingEnv is an abstract environment which is defined to support all kinds of trading environments. StocksEnv is simply an environment that inherits and extends TradingEnv.

4.2 Trading Actions

Upon searching on the Internet for trading algorithms, you will find them using numerous actions such as Buy, Sell, Hold, Enter, Exit, etc. A typical RL agent can only solve a part of the main problem in this area. When working in trading markets, one learns that deciding whether to hold, enter, or exit a stock is a statistical decision depending on many parameters such as budget, pairs or stocks to trade, money distribution policy in multiple markets, etc. It's a massive burden for an RL agent to consider all these parameters and may take years to develop such an agent.

These actions just make things complicated with no real positive impact. In fact, they just increase the learning time and an action like Hold will be barely used by a well-trained agent because it doesn't want to miss a single penny. Therefore there is no need to have such numerous actions and only Sell=0 and Buy=1 actions are adequate to train an agent just as well.

4.3 Agent

The goal of the agent here is to predict whether it should buy or sell stock at a particular price point. The data gives us history and the agent is able to use that history to take an appropriate action. This way, the agent is able to perform a limited number of actions and is able to take those actions effectively.

5 Implementation of System

5.1 Dataset for Stocks

The dataset used is the Nifty Indices Dataset from Kaggle. The trading data is shown at a day level This is a dataset containing the index prices of the 50 stocks that together are considered to represent the state of the NSE. The data contains Open, High, Low and Close Prices from 2000 to 2021, i.e. 22 years worth of financial data. The columns are as follows:

1. Date: Date
2. Open: Price from the first transaction of a trading day
3. High: Maximum price in a trading day
4. Low: Minimum price in a trading day
5. Close: Price from the last transaction of a trading day
6. Volume: Number of units traded in a day
7. Turnover: Closing price adjusted to reflect the value after accounting for any corporate actions
8. P/E: Number of units traded in a day
9. P/B: Number of units traded in a day
10. Dividend Yield: Number of units traded in a day

5.2 Dataset for Cryptocurrency

The dataset used is the Ethereum USD Dataset from Kaggle. The trading data shown is at the day level. The columns are as follows:

1. Date: Date
2. Open: Price from the first transaction of a trading day
3. High: Maximum price in a trading day
4. Low: Minimum price in a trading day
5. Close: Price from the last transaction of a trading day
6. Adj Close: Closing price adjusted to reflect the value after accounting for any corporate actions
7. Volume: Number of units traded in a day

5.3 **A2C**

In actor critic methods, The “Critic” estimates the value function. This could be the action-value (the Q value) or state-value (the V value). The “Actor” on the other hand, updates the policy distribution in the direction suggested by the Critic (such as with policy gradients). Both the Critic and Actor functions are parameterized with neural networks. In the derivation above, the Critic neural network parameterized the Q value — so, it is called Q Actor Critic.

The Advantage Actor Critic has two main variants: the Asynchronous Advantage Actor Critic (A3C) and the Advantage Actor Critic (A2C). A3C was introduced in Deepmind’s paper “Asynchronous Methods for Deep Reinforcement Learning” (Mnih et al, 2016). In essence, A3C implements parallel training where multiple workers in parallel environments independently update a global value function—hence “asynchronous.”

One key benefit of having asynchronous actors is effective and efficient exploration of the state space. A2C is like A3C but without the asynchronous part; this means a single-worker variant of the A3C. It was empirically found that A2C produces comparable performance to A3C while being more efficient.

5.4 **DQN**

In Q-learning, a memory table Q[s,a] is built to store Q-values for every possible combination of s and a (which denote the state and action, respectively). The agent learns a Q-Value function, which gives the expected total return in a given state and action pair. The agent thus has to act in a way that maximizes this Q-Value function.

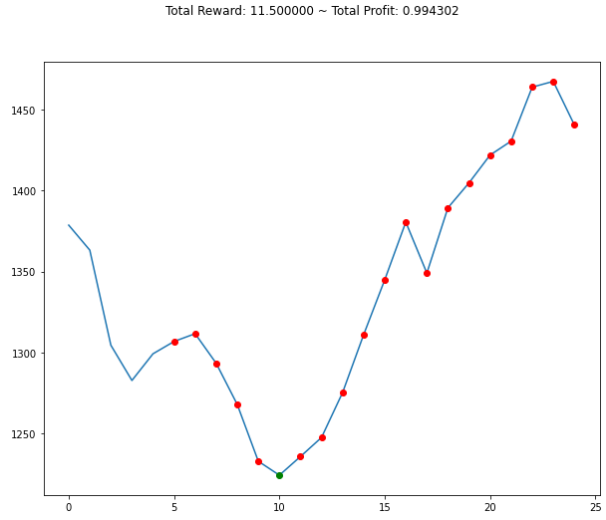
The memory and computation required for the Q-value algorithm would be too high. Thus, a deep network Q-Learning function approximator is used instead. This learning algorithm is called Deep Q-Network (DQN). The key idea in this development was thus to use deep neural networks to represent the Q-network and train this network to predict total reward.

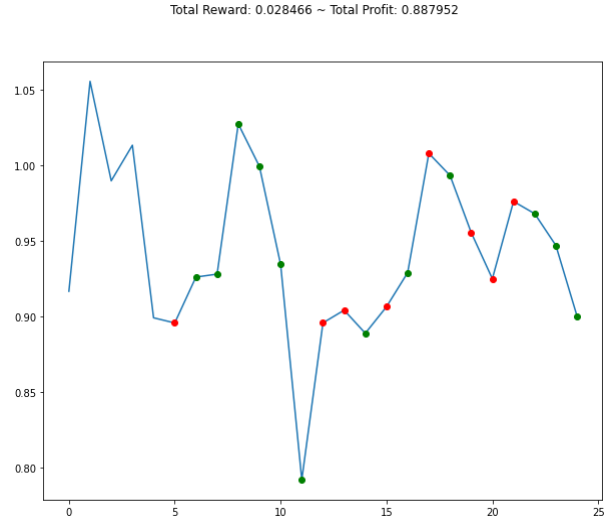
6. Results and Discussion

We have created 4 models in total which perform various tasks individually so to summarize we have two domains, Crypto and Stocks where we have trained the Advantage Actor Critic model and Deep Q Network. They are trained on NIFTY dataset and Ethereum USD dataset and have generated the following observations.

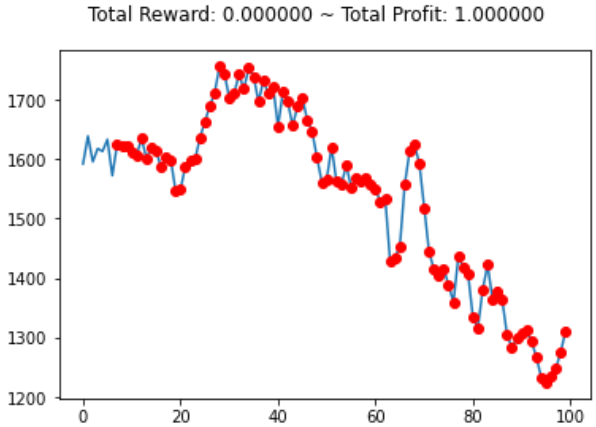
Output of each model on the Given Stock Value of the assets

Crypto A2C	Stock A2C





Crypto DQN	Stock DQN



Metrics		Crypto Trading		Stock Trading	
		A2C	**DQN**	**A2C**	**DQN**
Random	**Reward**	-0.075	-0.069	28.21	-150.54
	Profit	0.76	0.86	0.73	0.630
Trained	**Reward**	0.028	0	11.5	0
	Profit	0.887	1.0	0.994	1.0

As we can observe above the model when starting with random weights and no experience in the environment has performed very poorly for both the architecture but after training the A2C has better reward compared to DQN but also has less profits compared to DQN. This pattern is reflected in both the cases.

7. Conclusion and Future Work

In Conclusion we have created four models which have positive profit generation capabilities and can be used to do trading in the Crypto market and in the stock market. The models may not be very efficient in their strategies that they employ to generate these profits but this can be improved upon by researching and trying other model free agents and techniques. We can also try to train these models on live data. We can also transfer the weights for the current NIFTY and etherium model to other profiles by applying transfer learning. For the architect of the model used in DQN we can use different types of Deep Learning Architures such as CNN-LSTM models which would allow the agent to take context dependent decisions and hopefully have mode Educated decisions for the given stock value.

8. REFERENCES

- Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research. 2021.
- Jiang Z, Liang J. Cryptocurrency portfolio management with deep reinforcement learning. In 2017 Intelligent Systems Conference (IntelliSys) 2017 Sep 7 (pp. 905-913). IEEE.
- Lee JW. Stock price prediction using reinforcement learning. InISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570) 2001 Jun 12 (Vol. 1, pp. 690-695). IEEE.
- Wu X, Chen H, Wang J, Troiano L, Loia V, Fujita H. Adaptive stock trading strategies with deep reinforcement learning methods. Information Sciences. 2020 Oct 1;538:142-58.
- Bisht K, Kumar A. Deep Reinforcement Learning based Multi-Objective Systems for Financial Trading. In2020 5th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE) 2020 Dec 1 (pp. 1-6). IEEE.
- Chakole JB, Kolhe MS, Mahapurush GD, Yadav A, Kurhekar MP. A Q-learning agent for automated trading in equity stock markets. Expert Systems with Applications. 2021 Jan 1;163:113761.

APPENDIX

```
import gym
import gym_anytrading

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam

from rl.agents import DQNAgent
from rl.policy import MaxBoltzmannQPolicy
from rl.memory import SequentialMemory

import numpy as np
import pandas as pd


# %% [markdown]
# ## Check dataset

# %%
df=pd.read_csv("../data/ETH-USD.csv")
df.set_index("Date",inplace=True)
df.head()

# %% [markdown]
# ## Make OpenAI Gym Environment

# %%
# Passing the relevant data to gym and creating our environment

env=gym.make('stocks-v0', df=df, frame_bound=(90, 110), window_size=5)
states = env.shape
actions = env.action_space.n

# %%
# running the test environment

state=env.reset()
```

```
while True:
    action=env.action_space.sample()
    n_state, reqard, done, info= env.step(action)
    if done:
        print("Info: ", info)
        break

# %% [markdown]
# ## Model

# %% [markdown]
# ### Intializing

# %%
def build_model(states, actions):
    model = Sequential()
    model.add(Flatten(input_shape=(1,5,2)))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(actions))
    return model


model = build_model(states, actions)
model.summary()

# %%
def build_agent(model, actions):
    policy = MaxBoltzmannQPolicy()
    memory = SequentialMemory(limit=100000, window_length=1)
    dqn = DQNAgent(model=model, memory=memory, policy=policy,
                                    nb_actions=actions,  nb_steps_warmup=1000,
target_model_update=1e-1)
    return dqn

# %% [markdown]
# ## Training

# %%
env.shape
```

```
# %%
vals = 0
steps = 100000

model = build_model(states, actions)
dqn = build_agent(model, actions)
dqn.compile(Adam(learning_rate=1e-1), metrics=['mae'])

# %%

vals = dqn.fit(env, nb_steps=steps, visualize=False, verbose=1)


# %%
vals.history['episode_reward']

# %% [markdown]
# ### Testing

# %%
scores = dqn.test(env, nb_episodes=15, visualize=False, verbose=1)
print(np.mean(scores.history['episode_reward']))

# %%
# env = gym.make('stocks-v0', df=df, frame_bound=(90, 110), window_size=5)
obs = env.reset()

while True:
   obs=obs[np.newaxis,np.newaxis,...]
   output = model.predict(obs)
   # print(output[0][1])
   action, state=output[0][0],output[0][1]
   # print(action)
   obs, reqrds, done, info = env.step(action)
   if done:
       print("Info: ", info)
       break


# %%
dqn.save_weights('../models/crypto_dqn_weights.h5f', overwrite=True)
```

```
# %%
dqn.load_weights('../models/crypto_dqn_weights.h5f')

# %%
env.render_all()
```

```
            break
```

```
Info:  {'total_reward': 28.200000000000273, 'total_profit': 0.734396311320634, 'position': 0}
```

```
plt.figure(figsize=(10,8))
plt.cla()
env.render_all()
plt.show()
```