Introduction to Machine Learning Project

Reinforcement Learning Trading Algorithm

Joao (John) Ji Won Lee

December 15, 2021

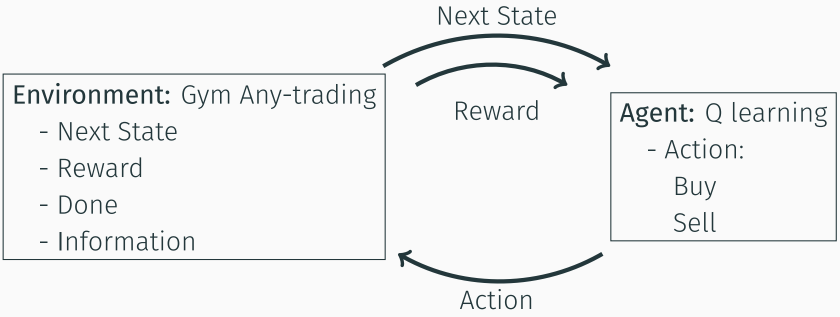
Net-ID: JJL720

Table of contents

Project

Project: Using reinforcement learning to create a trading algorithm for the Apple Stock. Specifically, the Q learning algorithm with the sequential model was adopted to create a better estimate of the Q-values.

Reinforcement Learning Diagram:



GYM Any-trading & Data-set

Gym Overview:

Gym any-trading is an environment that takes in actions and outputs rewards and the next state. Additionally, gym allows to customize the environment with a particular stock data.

Data:

The data used in this project was downloaded from yahoo finance using yahoo_fin package.

Data Details:

Stock: Apple

Ticker: APPL

Time Horizon: 12/04/2010 to 12/03/2019

Features:

Adj. Close	Open
Low	Close
Trading Volume	7 Days - SMA
High	

Reward

Gym Reward:(Problem)
Gym default reward is unknown. From my test, the reward was always positive regardless of the profit loss. The total reward increased more when there was a profit gain, but with a profit loss the reward was still positive.
Reward Improvement:
In order to improve the reward, I had to create my own reward function that would either reward or punish the agent for the given action at the given state. There were a total of three situations the rewards would need to be updated.

Position	Action	Reward Update
Add stock to portfolio	Buy	$next\_state_p - curr\_state_p = reward$
Sells stock from portfolio	Sell	$curr\_state_p - portfolio_p = reward$
Portfolio empty	Sell	Opportunity Cost = reward

Action

Gym Action:(Problem)

Gym allows for one action to either buy or sell. If the portfolio is empty and the action is to sell, gym will choose to not do anything. If the portfolio is not empty and the action is to sell, gym will sell all the portfolio at the given current state price. If the action is to buy gym will buy the stock and add to portfolio.

Action Improvement:

To improve environment a queue was imported. The queue appends the price of the stock when the agent chooses to buy. The queue pops the oldest price in the queue when the agent chooses to sell. Additionally, when the environment is at the last step the queue liquidates the portfolio at the given state price.

Action Improvement: Budget

To prevent the RL algorithm to excessively buy the stock. A budget and loan system was adopted. The environment allows for the budget to go negative as a loan with a 5% interest rate given that if the budget is bellow $-500 the loan interest rate increases to 11%.

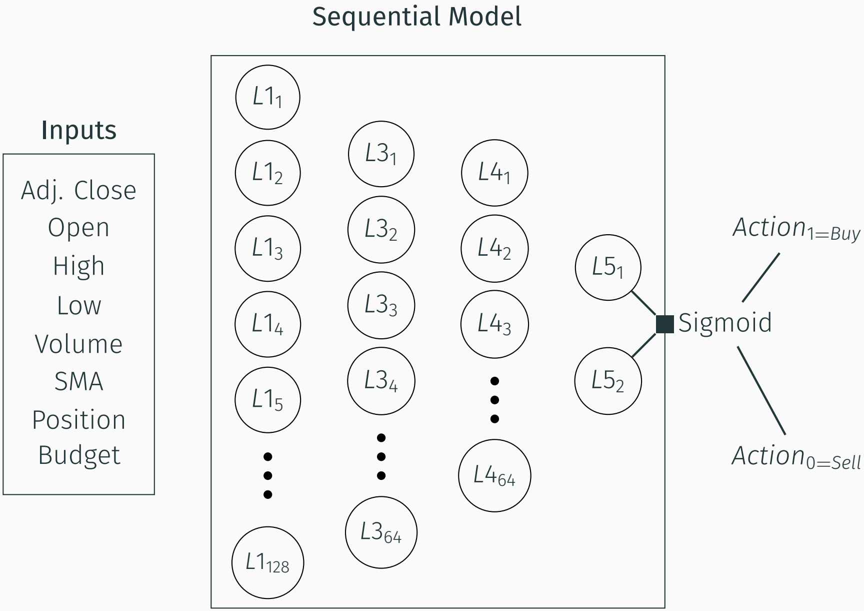
Q - learning

```
Initialize:
targets = numpy array
states = numpy array
exploration = between 0 to 1
$portfolio_{prices}$ = queue
for numbers or iterations do
    while True do
        Exploration and Exploitation of Environment
    end
    model.fit(states,targets)
    exploration -= 0.001
end
```

Q - learning Cont.

```
while True do
    Q-values$_{current\_State}$ = $model.predict(current\_state)$
    target[idx] = Q-values$_{current\_State}$
    if e-greedy random then
        action = random action
    else
        action = Highest of the Q-values$_{current\_State}$
    end
    reward,next_state, done, info= environment.action(action)
    if Action == Buy then
        portfolio$_{prices}$.enqueue(current_state$_{price}$)
        if budget >= current_state$_{price}$ then
            budget -= current_state$_{price}$
            reward = next_state$_{price}$ - current_state$_{price}$
        else
            budget -= current_state$_{price}$ (1 + loan_rate)
            reward = next_state$_{price}$ - current_state$_{price}$ (1 + loan_rate)
        end
    end
    if Action == Sell then
        if portfolio$_{prices}$ is empty then
            reward = opportunity cost = next_state$_{price}$- current_state$_{price}$
        else
            reward = current_state$_{price}$ - portfolio$_{prices}$.dequeue()
            budget += current_state$_{price}$
        end
    end
    Add position and Budget to the next_state
    targets[idx, action] = reward + learning_rate ( model.predict(next_state)[action] )
    states[idx] = log(next_state)
    idx +=1
    current_state = next_state
    if done == True then
        Break the while loop
    end
end
```

Sequential Model



Sequential Model Targets & Inputs

Inputs:Problems

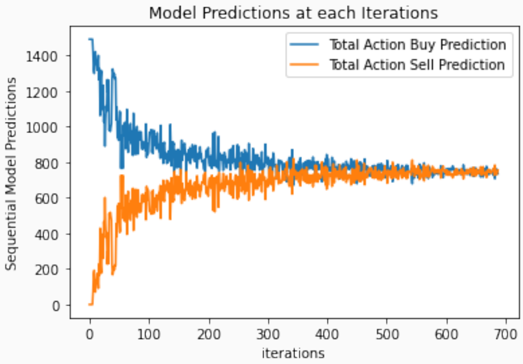
The inputs or stock features were too large. The price and the trading volume were very large numbers reaching the hundreds and thousands so the model was not able to properly train. To fix this problem, all the features were taken the log().

Targets:

The targets were updated based on the Q learning Bellman Equation

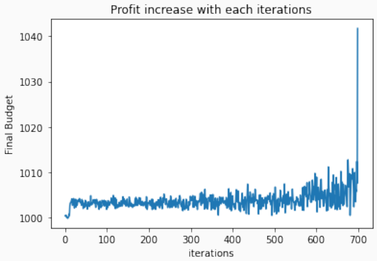
$$\underbrace{Target[s,a]}_{\text{Update Target[s,a]}} \mathrel{+}= \underset{\substack{|\\ \text{Learning-rate}}}{\alpha} [\underbrace{R(s,a)}_{\text{Reward}} + \underset{\substack{|\\ \text{Discount rate}}}{\gamma} \overbrace{Target'[s',a']}^{\substack{\text{Largest predicted target, given} \\ \text{new state and all possible actions} \\ \text{Predicted target with model.predict}}} - Target[s,a]]$$

Results - Training sequential model



The graph represents the sequential model being trained. The buy are the total buy predictions and the orange the total sell predictions. As the model is being trained, the buy and sell predictions become closer to 750 meaning at the sequential model begins to predict buying and selling.

Results - Profit



The graph represents the iterations of the profit during each training. This graph is not completely representative of the algorithms because it shows the profit during the training process. In the training process the exploration starts at a high level and decays with each iterations, so the actions the agent takes might not be perfect hence a lot of noise, but there is an upward trajectory indicating the algorithm is working.

Improvements

Reward:

To improve the reward and penalty, time value of money could be adopted. Additionally, it is important to understand that Amazon had an upward moving trend from 2010 to 2019, so it become easy for the algorithm to just buy and not perform any trades, so perhaps adopting the Sharpe ratio as part of the reward function can be helpful.

States Features:

Some Features that can be included in the states could be different holiday seasons or the performance of the stock at the given year(For example so stocks profits decreases during tax seasons). Also, adding a second stock to provide some risk hedging opportunities in the portfolio could be helpful in increasing the profits.