Application of Deep Reinforcement Learning in stock trading strategies

Introduction:

Artificial neural networks and deep learning are getting increasingly popular in every field due to their versatility and ability to extract complicated features from huge datasets. Stock prices are affected by a large number of factors which makes it very difficult to predict future prices. Since, deep learning specializes in predicting complex patterns, we can expect good results by using deep learning techniques in predicting stock prices.

Particularly, this project explores application of deep Q learning which is a variant of reinforcement learning to make decisions like buy, sell or remain neutral on behalf of a human in an environment like stock market.

Necessity of a trading strategy:

Trading strategy is a fixed plan which suggests when to buy, sell or hold stocks to ensure maximum profit over a period of time. Since the stock market is so dynamic, traders might get too greedy or too afraid and make trades at non optimal times. Without a certain strategy, even experts make mistakes by letting their human emotions get in the way of their logic. So, sticking to a strategy reduces or eliminates this emotional dependence and leads to an optimal profit.

Having a fixed strategy enables us to implement computer programs to make trades at the right time for us. Introducing computers to make these decisions can improve performance as they will make decisions quickly, without emotional interference.

Timeline for this Project

- **May 23rd - May 30th :** Getting Familiar with Python and some important libraries such as Numpy, Pandas, Matplotlib and Seaborn.
- **June 1st - June 8th :** Studied Basic Fundamentals of Reinforcement Learning such as Agent, States, Environment, Actions, Action-Values, Rewards, Policy, Bellman Equation, etc.
- **June 3rd - June 13th :** Assignment 1. Task - Take last 10 years weekly data for any 3 firms and predict their Adj Close price using any 5 ML models. Identify the best features that need to be used to make predictions more accurate. Show graphs to show feature dependency of Adj Close price. Also, show graphs of test and train data, predicted values and error values for using different parameters using the same model. Determine the accuracy of each model along with error metrics (like MAE, RMSE, MSE, etc.). Try to explore ML models as much as possible.
- **June 14th - June 18th :** Studied about DQN and its requirements and Read some papers to get some in-depth knowledge about how things are actually working.

- **June 19th - June 20th :** Mid - Term Evaluation for checking the overall progress in project work.
- **June 21st - July 19th :** Assignment 2. Task - Given the stock price till the (t-1)^th day, predict whether the stock price would go up,down or would remain sideways on the t^th day. So the current state is the close price list till the (t-1)-day and the number of stocks we have purchased on this time stamp.

  Possible actions would be:
  1. Buy the stock(which means we predict the stock price to go up)
  2. Sell the stock(which means we predict the stock price to go down)
  3. Hold on the position (which means do nothing)

  Given Variables along with conditions associated with them.
  1. MAX_TRANSACTIONS = k, maximum number of time we can do a buy after a selling
  2. TOTAL_MONEY = k2 , total money that we have in cash(virtual)
  3. CURRENT_STOCKS_BOUGHT = k3 , the number of stocks we have bought currently
  4. CURRENT_TRANSACTION_COUNT = k4, number of buys done before selling - You can only do the buy if CURRENT_TRANSACTION_COUNT<MAX_TRANSACTIONS

  Whenever we do a buy the amount with which we buy the stocks are x= TOTAL_MONEY/(MAX_TRANSACTIONS- CURRENT_TRANSACTION_COUNT) and the number of stocks bought in this buy signal = x/STOCK_PRICE at that moment.
  Whenever we do sell just increase your TOTAL_MONEY by CURRENT_STOCKS_BOUGHT*STOCK_PRICE and do the CURRENT_TRANSACTION_COUNT=0
  Now when to do a buy or when to do the sell:
  1. Do the buy whenever there is a buy signal and when CURRENT_TRANSACTION_COUNT<MAX_TRANSACTIONS
  2. When there is a sell signal from DQN , just sell any stocks if we had purchased if any
  3. When there is a hold signal , do nothing

  Reward: It is the profit or the loss that happens in any transactions
  DataSet: Use the 2009 January-2017 December data for training and then use the 2018 January-2019 December data to do the test(and report the profits made in it).

Theory

Q-learning:

Q-learning is a reinforcement learning algorithm that aims to determine best action in the given state. For this we create a table of size = No. of state * No. of actions which tell us how good an action ‘a’ is given we are in state ‘s’. We begin by filling the values of q-table by 0s or some other initialization method. Later we use these values to select the best action. We are in state ‘s’ then we take action ‘a’ and receive reward ‘r’. Based on these values our q-table is updated. We use the bellman equation(given below) to update the q-values.

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

While we are on it let's take a look at the exploration vs exploitation problem. If our agent does not explore then it will not be able to find an optimal strategy and if it keeps on exploring then there is no use of that optimal strategy. To tackle this problem we use a method called 'ε-greedy method'. In this method with probability ε the agent chooses a random action else it takes the greedy action. This method works fairly well and all we need to take care of is that ε decays to 0 in the end.

Limitations of Q-learning:

On paper Q-learning is a fantastic method. It works fairly well when the number of states and actions are limited but once either of them start growing, storing values in a table becomes infeasible. A simple example would be modelling a chess game using q-learning. The number of states goes over 10^{60}. If this seems doable then think about a problem where the states are continuous. In that case creating a table would be impossible and hence we need something different and that deep q-learning is a great candidate for that.

Deep Q-learning:

If we can get rid of the storage issues associated with q-learning then we can extend it to continuous state problems too. Deep Q-learning addresses this by using neural networks as function approximators and removing the need of tables altogether. We know neural networks are really good non-linear function approximators. We feed the network the current state and it spits out the q-values of all the actions. From this we can select the best possible action. But there is a convergence problem with deep q-networks and in order to tackle that we use 2 methods -

1. Replay memory
2. Target and Local network

Replay memory is basically a list of samples of (State, Action, Reward, Next State) at various iterations. We take this tuple and train the local network on this. Note that we do not train target network on this. After a couple thousand iterations we copy the weights of the local network to the target network. These 2 methods combined address the issue of convergence.

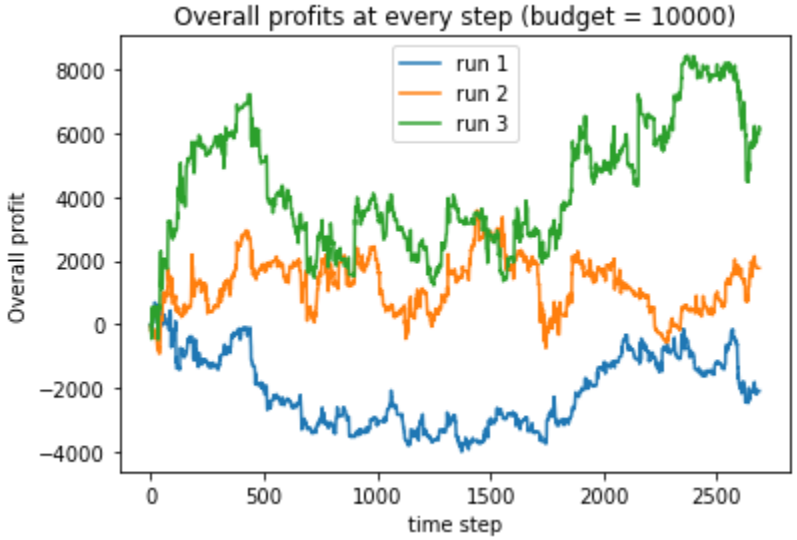
Stock trading and DQL overview:

Stock market is very dynamic and every moment in the stock market is different. Every time, the parameters that define a particular state will have different values. Stock prices also depend on that particular company's reputation at that particular moment in time, company's policies, and related government policy changes. It leads to a very huge number of parameters which are very likely to get repeated. Which means there are infinite states and no state is repeated again. In this context, Q learning becomes impossible because of the reasons mentioned in limitations of Q learning. Deep Q learning to be able to map state variables to action space using the deep Q network. Since DQN (deep Q network) is a function, it can be trained to approximate Q values of all possible actions from state variables by using historical data.

Setting up environment:

An environment class is prepared to simulate stock market conditions along with giving reward for every action an agent takes. This class can be initiated by giving historical data of a stock, initial budget, and some other required parameters. It has methods to take an action like buy, sell, or remain neutral. It will keep track of the budget, overall profit made and other parameters necessary to maintain the environment.

The environment class is tested by taking random actions at every time step and three episodes are simulated. We expect profit would be random and different in three cases. Graph of overall profits made at every time step in each episode is given here.



As we expected, in three different runs, at the final time step, profits made are different.

Setting up agent:

We first defined some attributes of the agent class which included variables like - epsilon, minimum value of epsilon, factor by which epsilon should be decreased, discount factor, inventory in which we store number of stocks in possession, etc.

Then we defined the network method which created our local and target network, the replay memory method which adds (State, Action, Reward, Next State) to the memory, action method which follows the ε-greedy policy, buy, sell and hold actions and the train method in which we actually train the agent with our data.

Issues Faced while Implementation and its Correction

Only One or Two Signal Generation leading 0 profit:

We started with a reward system where the agent gets 0 reward for buying and holding, however gets a variable reward (positive reward for profit and negative reward for loss) for selling the stocks. But this reward system resulted in generating only one signal for holding everytime and resulting in zero profit.

Possible Reason Identified :
In the training of model, our model is basically learning what action needs to be taken and what should be avoided based on the current state. In this process, our model was taking the action of selling the stocks randomly in the start which could have resulted in some negative rewards due to loss generated. Since the other two actions had only 0 reward associated with them, the q-values associated with them became greater than that of selling. Hence, as the value of epsilon decreased to a significant amount, our model started to take only two actions - buy and hold. As profit is only generated on selling of stocks, our model started to give 0 profit. Also, we have another condition on buying of stocks i.e. current transaction count should be less than max. Transaction count (constant integer chosen at the start) which stopped our agent to give a signal for buying after some time.

Changes made to overcome:
To avoid hold signal for everyday, we started to give some negative rewards to our agent whenever there is a chance to get some profit by selling but our agent signals for hold. Also, gave some positive reward to agent for signaling hold option at correct moment.

Outcome :
These changes failed to give the desired results as our model was even opting for the hold option after some training.

Possible Reason Identified :
The changes made in the reward system were not comparable to the negative rewards obtained on selling, so there was indeed a requirement to change the rewards accordingly.

Changes made to overcome:
This time we increased negative rewards for the hold option and reduced positive rewards for it. We also started to give positive rewards for buying option at appropriate moments and small negative rewards whenever there is an opportunity to obtain some profit and our model signals for buy. One last change was made to limit the negative rewards on selling of stocks (e.g. negative reward on selling should not be greater than a certain value). This ensured that our agent would not try to completely ignore action for sell which is the most important action for us to generate some profit.

Outcome : This made our agent to signal for all actions optimally to generate some profit over training the model.

Results:

The training process of the model takes about 10 minutes when trained for 50 episodes taking the size of the state vector to be 50.

The developed model was trained on the 2009 January-2017 December data procured from the Yahoo Finance Website and was then used to predict the stocks and perform transactions on the 2018 January-2019 December data. Starting with an amount of 10000 USD, the model was able to make a profit of about 870 USD in this span of 2 years.

Conclusions:

Looking at the way our DQN model performed before and after refining the rewards associated with different State-Action pairs, it is clear that the definition of the immediate rewards corresponding to different State- Action pairs and some appropriate bounds on the values of the rewards plays a very crucial role in the way the Model learns and performs on the real data-set.

In order to refine the Reward function as well as all the model parameters, it is best to take a recursive approach where the initial definitions and values are chosen intuitively and then the model is allowed to train and after that the reward functions and parameters are adjusted accordingly to repeat the same process till we are finally satisfied with the results.

The DQN seems to be a promising technique to apply to solve the rather complicated problem of devising Stock trading strategies in an environment where the dynamics of stock prices is governed by numerous factors which often remain uncaptured by humans.