Auto Stock Trading- Reinforcement Learning

Abstract:

In the project "Auto Stock Trading- Reinforcement Learning" we made an agent which we call RL agent (Reinforcement Learning Agent) that can predict the flow of the stock market and based on the inputs and prediction, the agent takes an action that it believes can maximize the reward. The basic idea behind working on the project is that every day we get new information which is considered as the state and based on the provided information we take an action like buying, selling, or holding the share every day (time step). And after every action, we arrive at the next stage and obtain a reward. The environment for this project is the stock market itself and the actions taken in the environment will affect the environment. This project is a simulation and we have used historical data of Apple, Motorola, and Starbucks our aim in this project was to build an environment object that stimulates the stock market.

Introduction:

A share/ stock market is the aggregation of the buyers and sellers of shares or stock, and these shares/ stock represent ownership claims in a business. In general, our goal in a stock market is to make money, this goal can be achieved if we invest using a certain strategic investing because stock trading is a sequential task. Therefore, we can use Reinforcement Learning in the stock market because Reinforcement Learning optimizes sequential tasks such as stock market trading. With the help of reinforcement learning, we can map states to value which is the expected future return i.e., it can look at the state of information today and select the best action that can maximize our expected future return.

The other way by which we can do this is by using machine learning in the stock market. Machine learning can be applied to the stock market as well, but it is confined to predicting the value of a stock or even just the direction of whether the stock will go up or down i.e., we can map the trend of a stock. But this type of information barely makes any difference because ultimately, it's the human who needs to make the trade. And there can still be some doubts regarding whether to trade or not because there may be a slight possibility that the model predicts the direction of the stock

will go up but after that, if the stock direction goes down sharply then purchasing the stock will be of no use. This is the key difference between supervised and unsupervised learning versus reinforcement learning. In supervised learning, we get a prediction, and we must take a decision based on the prediction. In contrast, a Reinforcement learning agent takes an action which should believe that it will maximize our reward.

Working:

To simulate the environment of a stock market we have instantiated an environment object which is MyTradingSimulationEnv(), and then we initialized a Boolean done flag which is set to false. To return to the initial state i.e., the starting position of the environment we need to call env.reset(). We then enter into a while loop, which will quit if and only if the done becomes true inside this while loop. The next step is to actually perform the action in the environment, and we do this by calling env.step(action), passing the action in env.step() will return a few things. The first thing it returns is the next state. The Second thing it returns is the reward for arriving in the next state, and the third thing it returns is a done flag to tell us whether or not the episode is over. And finally, it returns an info dictionary that can give additional information about the environment, and we populate this info dictionary to tell us the current value of the portfolio. In the end, the next day variable is assigned to the state variable in the case where on the next state the agent needs to use the state to choose an action. The state is a vector that consists of three parts, how many shares of each stock are owned by you, the current price of each stock, and how much-uninvested cash is available.

In the coding part, the first function we create is the get_data() and in this get_data() function, we import the data stored in CSV which we can load into the data frame, and to simplify the problem we used closed values only. In the CSV file, we have defined the share of Apple as AAPL, the share of Motorola as MSI, and the share of Starbucks as SBUX, each row in data corresponds to the same day and for each stock, we have data from February 2013 to February 2018. The next function which we create is the get_scaler(env) function and this function takes the environment object because it will be used in fitting the scaler, the basic idea is we must have some data to get the right data for the scaler parameter. And when we choose an action to perform the only thing we are required to do is sample a value from the action space. When we perform all these steps,

we then create a standard scaler object called scaler and we fit that into states by scaler.fit(states), and in the end, we return this scaler object. The next object we create is maybe_make_dir(directory) and this function is more than a simple utility function because it checks if a particular directory exists or not and if it does not exist then it creates one. The next thing we have is our LinearModel class and in the constructor, we take two arguments, the input size, and the output size, and we initialize our weight matrix and biased matrix to be a random matrix. We also create instance variables vW & vB to store momentum and we set them to zero. This is followed by a prediction function predict(self, X), which takes a 2-dimensional array of size X. The function that performs a major operation in the function is the sgd function as it performs one step of gradient descent, in this function we take input arguments and target X & Y along with these we also take the learning rate and momentum. After all the mathematical calculations we obtain 2 functions load_weights and save_weights and these function load and save the model.

We then define a class for our environment and we name it as MultiStockEnv. In this class, we have a constructor that takes two arguments, next we have the Constructor which takes in two arguments a two-dimensional data array containing a time series of stock prices and also the initial investment amount with a default value of twenty thousand dollars. In this, we set the attribute stock price history, and initial investment and we initialize a few attributes to none. Next, we set the attribute action space which is equal to the integers 0 up to 3 to the power and stock next for convenience will want to have a corresponding action list. Since we have twenty-seven possible actions this will be a list of length twenty-seven. Finally, we call the reset function which initializes is a few more attributes and returns the initial state, inside this function we set the attribute current step equal to zero which means that we point to the first day of the stock price in our dataset. Next, we have an array that gives information about how many shares of each stock we own so when we start, we are not going to own any stock. We then have the stock price which tells us the current price of each stock on the current day. This is just the time series index by the current step. To begin the investment, we use the cash-in-hand attribute. The step function then acts as the environment and returns the next state and the reward. We then increment the current step pointer and also the current stock prices attribute so the next day the price goes up to the price for the next

day. Lastly, we return the next day, reward, done flag, and information directory. To perform the trading, we use the trade function.

For the Reinforcement Learning Agent, we create class DQNAgent, this agent is responsible for taking experience learning and utilizing it to maximize the future reward, the constructor here takes the size of the state and the number of actions as input. This is then followed by an act function that takes a state and uses epsilon greedy to choose an action based on the state. The function by which the agent learns from the data is the train function and this is the function where we create our supervised learning dataset. In the end, we update epsilon to reduce the amount of exploration over time.

In the main section, there are configuration variables of the model, rewards, number of episodes, batch size, and initial investment. We then create an argument to run the command line argument, after this, we create a model directory and reward directory in the case where they do not exist. To fetch information about the time series we create a getData function. We split the data into train and test, the first half is train and the second half is a test after which we set an attribute for some functions as well. After performing all the required operations, we will save the reward we got from train data in train.npy and test data in test.npy.

Conclusion:

Considering that our data set is from daily closed prices in the span of five years since the test period is half of the data set which means this increase is over a span of about 2.5 years. The rewards we got from the training data are quite good, we can even get up to a 2.5 times increase in our original investment which is extremely high. The result for testing is also quite good, our original investment usually always increases by a significant amount.

A Report by,
Akash Yadav, ID: 301426295
Ritvik Gaur, ID: 301426477
Raghav, ID: 301390675