Kshitij Chhajed

Deep Reinforcement Learning

10-Dec-2021

1. Overview

The task of this assignment is to implement Reinforcement learning using Q-learning algorithm. The project aims to devise a strategy such that the algorithm can learn the trends of the stock prices. It targets to perform a series of trades over a period so that user can end in profit with a goal of $120000 total account value for $100000 of initial investment.

2. Dataset

The assignment is based on historical stock data for Nvidia dated from 10/27/2016 to 10/26/2021 with a total of 1258 entries. The parameters highlighted in this data are stock opening and closing prices, intraday high and low, traded volume and adjusted closing. This dataset has been included in the 'NVDA.csv' file.

3. Languages/tools used

I have used Google Colab for implementation. I have used below libraries:
numpy – to perform numerical operations on matrices
matplotlib – to plot and display graphs between various parameters
pandas – to perform operations efficiently on the data from the dataset
math – to access common math functions and constants
gym, spaces – Gym is used to import environments which can be used to develop and compare RL algorithms. Spaces from gym is used for easy declarations and operations on different spaces like observation_space, action_space, etc. necessary for creating the environment

4. Environment

The Environment has been setup with the help of gym library and its sub-libraries. The important aspects of the environment could be described as below:

A. Agent:

It is defined as the learner or the decision-maker of the environment.

B. State:

State defines the current situation of the agent in the environment.
The states have been derived from the observation vector which has been defined as:
observation = [price_increase, price_decrease, stock_held, stock_not_held]
observation [1, 0, 0, 1] corresponds to state = 0
observation [1,0,1,0] corresponds to state = 1
observation [0,1,0,1] corresponds to state = 2
observation [0, 1, 1, 0] corresponds to state = 3

The meanings of the above observation vectors and their corresponding states can be summarized as follows:
0 - Price of the stock has increased and the agent doesn't hold any shares.
1 - Price of the stock has increased and the agent holds shares.
2 - Price of the stock has decreased and the agent doesn't hold any shares.
3 - Price of the stock has decreased and the agent holds shares.

C. Actions:

Action is what the agent can do in each state.
There are 3 actions defined by the action vector as below:
0 – Buy – Directs the agent to buy shares
1 – Sell – Directs the agent to sell off existing shares
2 – Hold – Directs the agent to hold current position

D. Goal:

The general aim of a Reinforcement Learning algorithm is to maximize its cumulative reward. In our project, end goal of the agent is to maximize the profit with a target to achieve $120000 in the total account value.

E. Reward:

It is defined as the feedback given by the environment for the action taken by the agent in a particular state. Negative values of reward are generally considered as punishment/penalty. For the given environment, the reward has been defined in same manner for training as well as for evaluation.

a. When action = Buy –
   If agent already has shares, a penalty valuing '-10' is applied. If the no. of shares to be bought are positive, the reward is calculated as reward = 1+penalty. If no. of shares to be bought are less than or equal to 0, reward = -10.

b. When action = Sell –
   If agent has bought the shares at a positive value (meaning agent already holds the shares), reward is calculated using the formula:
   Reward = ((selling value – booking value)/booking value) * 100
   If buying price is negative (meaning agent doesn't hold the shares), reward = -10.
c. When action = Hold –
   If agent the bought the shares at a positive value, reward is calculated using the formula:
   Reward = ((current stock value – booking value)/booking value) * 100.
   If the above condition is not met, meaning the agent does not hold the shares, a reward =-1 is supplied.

The Environment is built using a class with 4 functions namely init(), reset(), step(), render().

a. Init():

This function initializes all the variables necessary to define the environment and perform subsequent operations and actions on it. Firstly, it reads the dataset file and splits the data into training and testing parts. It later defines the observation vector space and the action vector space. It also defines the maximum timesteps to be considered while training and evaluating data. It holds a Boolean variable 'train' which defines whether the task is to train the agent or evaluate. It also defines the no. of days for which the agent considers the stock in 'number_of_days_to_consider' variable.

b. Reset():

It initializes all variables to values needed at the beginning of training/testing the agent. It returns the observation vector by calculating whether the price has increased or decreased over the time period (number_of_days_to_consider variable) being considered by the agent. In the provided example, since the no. of days to be considered =10, the reset() method counts price_increase =1 (and price_decrease =0) if ratio of sum of increase in price/10 remains greater than 0.5; otherwise vice-versa.

c. Step():

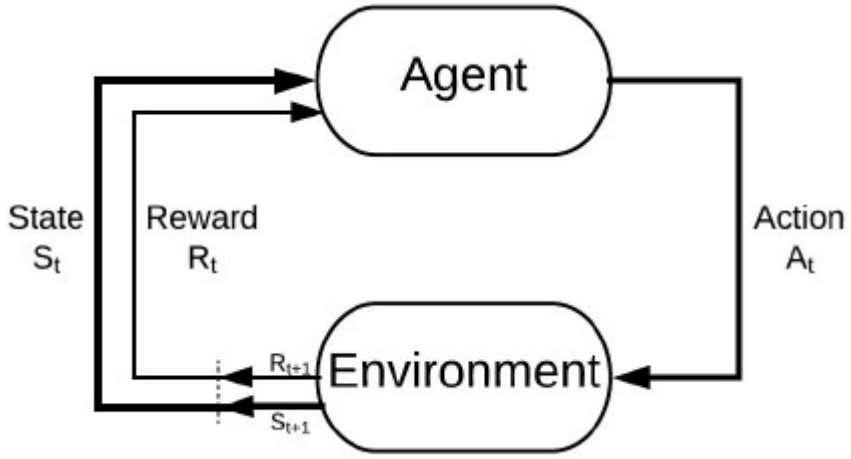
This method calculates all the rewards for every (state, action) pair and gives the updated states (observation) vector as a return value. It requires the action taken by the agent as the input. It hosts a Boolean variable to indicate whether or not the episode (iteration to make agent reach the terminal state) has completed.

d. Render():

This method is used to display the agent's total account value over the provided duration.

5. Q-learning

Reinforcement Learning is the science of making the best judgments possible based on past experiences. The goal of RL is to maximize an agent's reward by performing a series of behaviors in response to a changing environment.



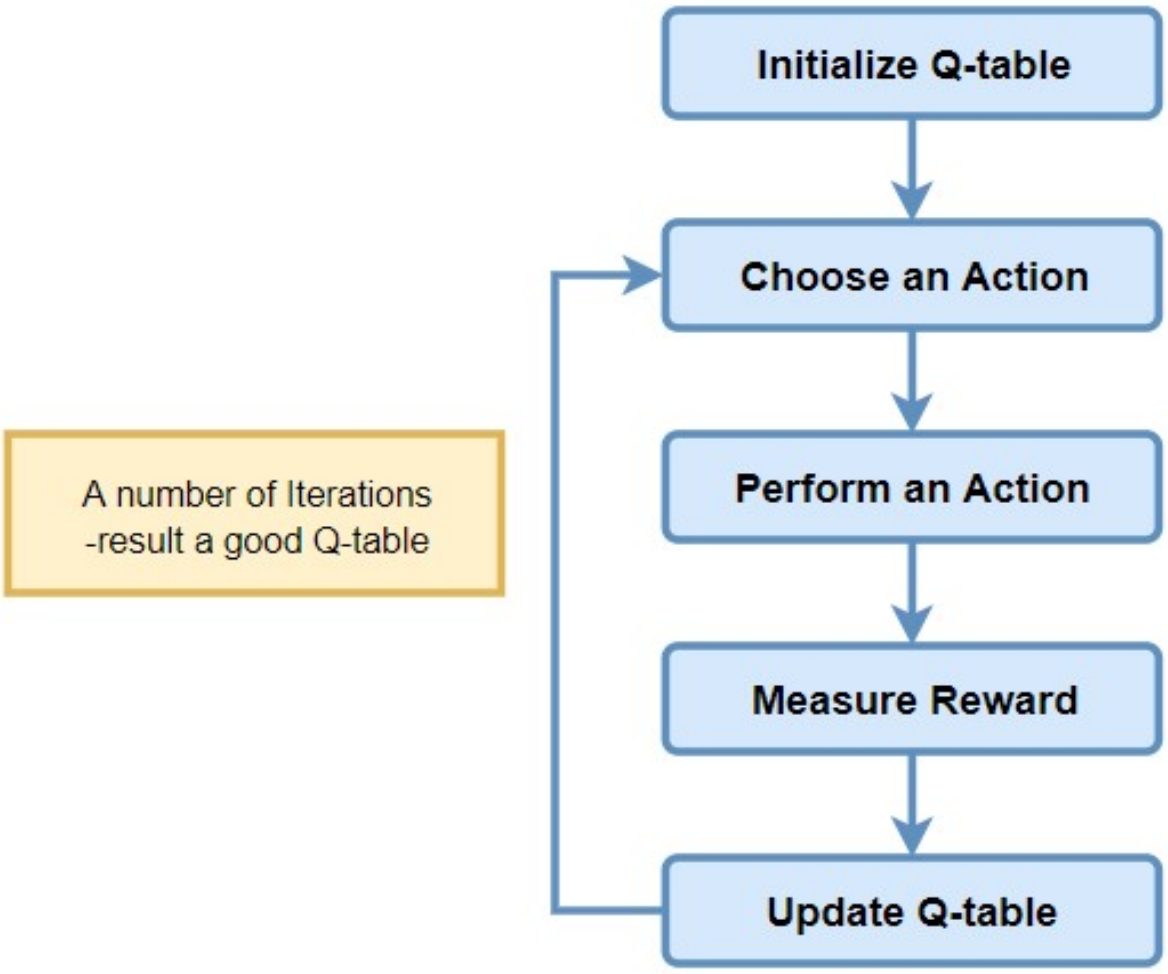
Q-learning is a reinforcement learning algorithm that does not require a model. Q-learning is a learning algorithm that is based on values. Value-based algorithms use an equation to update the value function (particularly Bellman equation). The alternative kind, policy-based estimation, uses a greedy policy obtained from the previous policy improvement to estimate the value function. The letter 'Q' stands for quality in Q-learning. Here, quality refers to how beneficial a specific activity is in obtaining a future reward.
Important hyperparameters and their description:

Alpha –
The learning rate or step size is defined by alpha. Alpha is a real number that ranges from 0 to 1. The agent learns nothing from new actions if alpha is set to zero. If we set alpha to 1, on the other hand, the agent entirely disregards prior knowledge and only values the most recent data. Q-values change more quickly with higher alpha values.

Gamma –
The discount factor is called gamma. Gamma is a real number that ranges from 0 to 1. When gamma is set to zero, the agent totally disregards future rewards. Such agents are just interested on the current rewards. If we set gamma to 1, on the other hand, the algorithm will look for high rewards in the long

run. Conversion may be hampered by a high gamma value: adding non-discounted rewards results in high Q-values.

Epsilon –
The epsilon parameter is connected to the Q-learning algorithm's epsilon-greedy action selection mechanism. If epsilon is set to 0, we never explore and instead rely on what we already know. Having the epsilon set to one, on the other hand, forces the algorithm to always take random actions and never use previous information. Epsilon is usually chosen as a tiny value close to 0.



6. Implementation

For training purposes, I have incorporated the concept of epsilon-decay. This parameter helps to avoid being stuck in a local minima by gradually decreasing the value of epsilon using a decay-rate. Higher the value of epsilon, higher is the tendency for the agent to use random-approach. As the epsilon value reduces, greedy-approach increases and randomness decreases which in turn reduces exploration and increases exploitation.

For calculating the Q-table values, I used the below formula:

$$Q(s,a) = (1\text{-alpha}) * Q(s,a) + \text{alpha} * (\text{reward} + \text{gamma} * \max_{a'} (Q(s',a')))$$

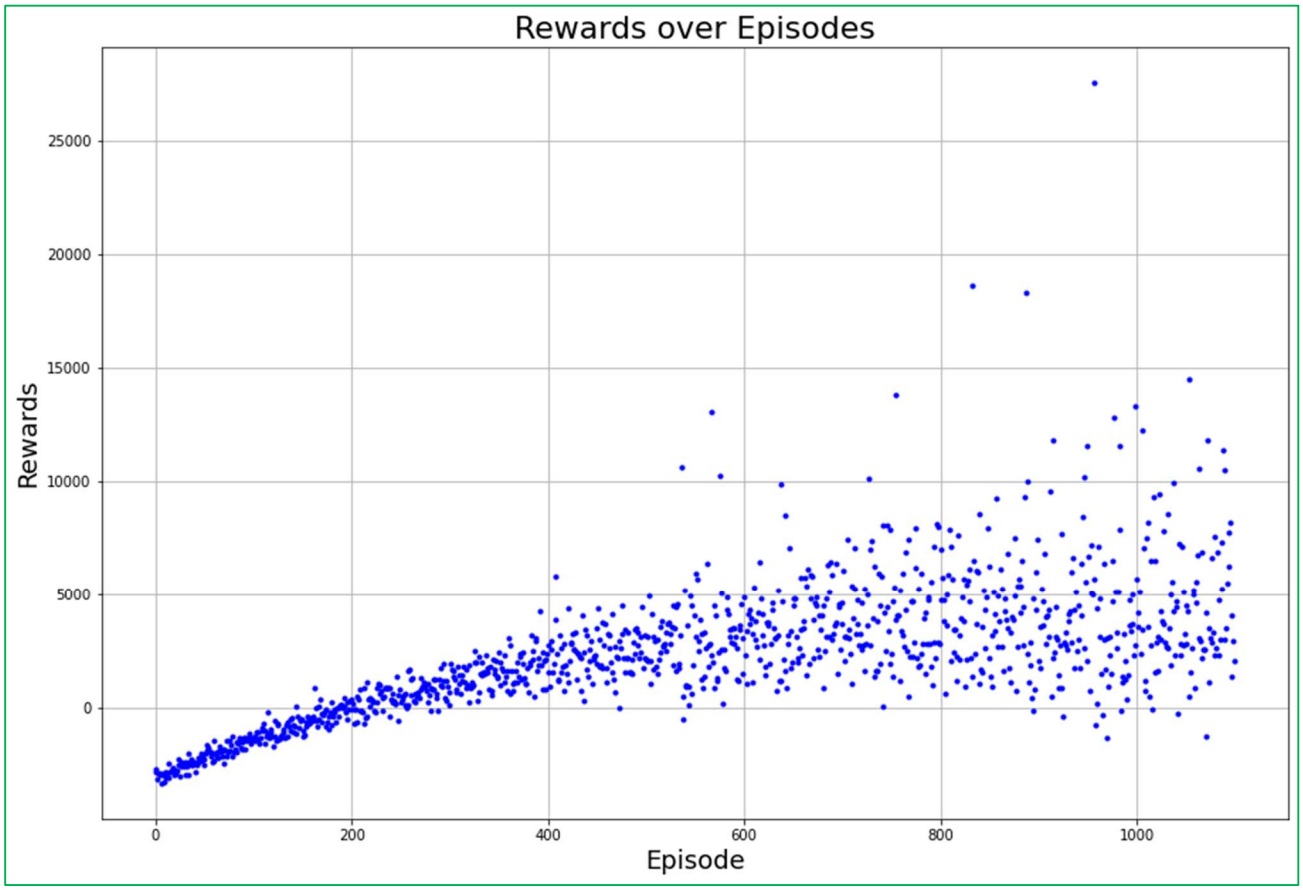
7. Training results

I have randomly tried different values of hyperparameters between the below ranges:
no. of episodes between 500 to 5000, alpha (learning rate) between 0.01 to 0.125, gamma between 0.85 to 0.99, epsilon between 0.9 to 0.99, decay_rate between 0.0009 to 0.05.
I could see the best results for the values provided as under.

Hyperparameters used for getting the results shared below:

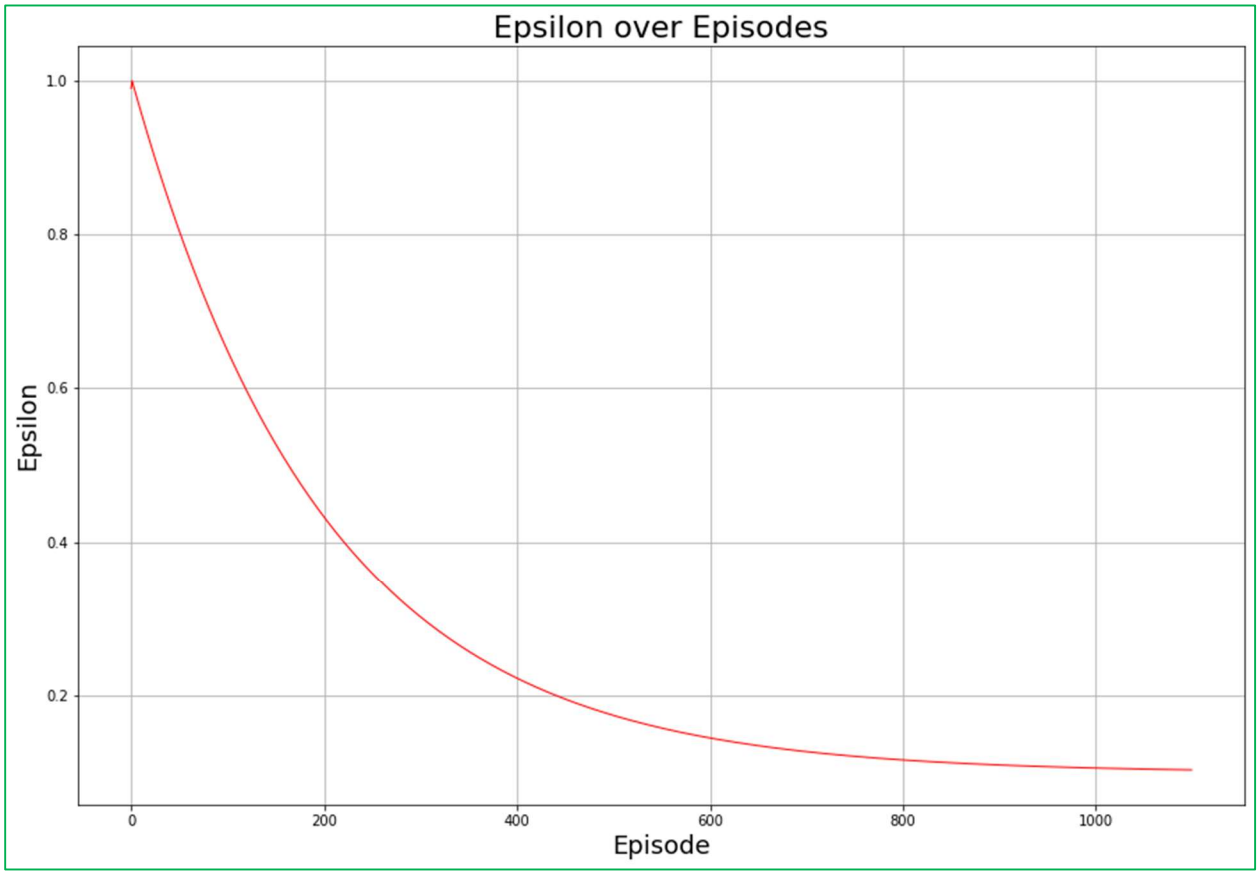
```
self.noofepisodes = 1100
 self.alpha = 0.11
 self.gamma = 0.99
 self.epsilon = 0.99
 self.epsilon_init = 1
 self.epsilon_end = 0.1
 self.decay = 0.005
```

Plot showing total reward per episode:



As we can see, the agent's reward is increasing over time gradually which indicates that it is being trained on the Q-learning algorithm properly.

Plot showing epsilon values per episode:



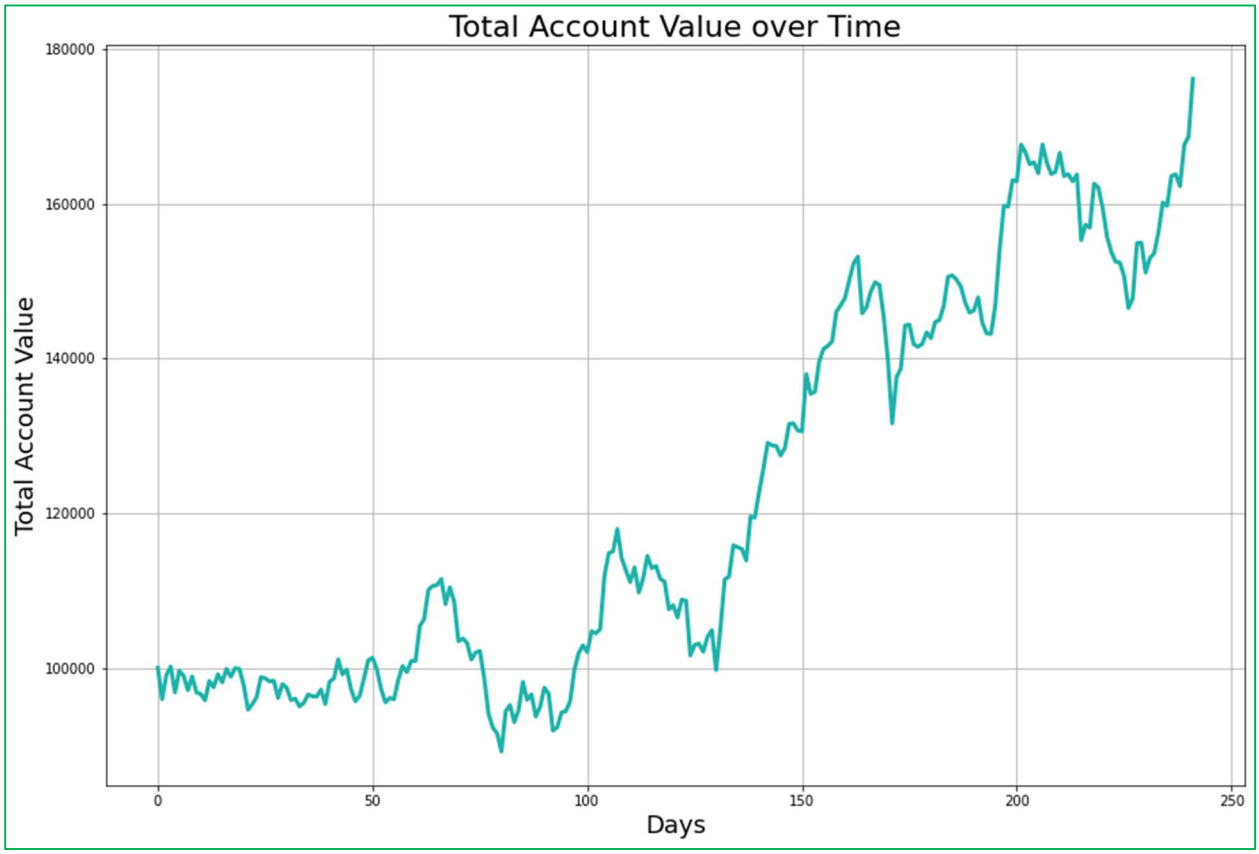
The epsilon value is being calculated using the formula below for every iteration (i):

$$Epsilon = epsilon\_min + (epsilon\_max - epsilon\_min) * e^{(-decay\_rate*i)}$$

8. Evaluation results

Once the agent has been trained, we can directly use the Q-table to determine what should be the optimal policy to achieve maximum profit. With the parameters used as described earlier, we could observe that the total account value for the agent reached around $175000. This leaves the agent with a profit of $175000 (current value) – $100000 (initial investment) = $75000

Agent's account value over time after it has been trained completely:



The agent crosses the target value of $120000 successfully as we can see from the above graph.

9. Credits

1. https://gym.openai.com/docs/
2. https://www.baeldung.com/cs/epsilon-greedy-q-learning
3. https://www.youtube.com/watch?v=EnCoYY087Fc&ab_channel=TheCodingLib
4. https://www.youtube.com/watch?v=WLOUElrYvyo&ab_channel=chris_mutschler
5. Piazza, Course slides
6. Numpy support docs
7. https://stackoverflow.com/questions/53198503/epsilon-and-learning-rate-decay-in-epsilon-greedy-q-learning
8. https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c