
PROJECT - 3

Sai Teja Cherukuri (50418484)

cheruku3@buffalo.edu

10th December 2021

---

1. Project Overview

The goal of the project is to learn the trends in stock price and perform a series of trades over a period of time and end with a profit. In each trade we can either buy/sell/hold. We will start with an investment capital of $100,000 and the performance is measured as a percentage of the return on investment. We will use the Q-Learning algorithm for reinforcement learning to train an agent to learn the trends in stock price and perform a series of trades.

2. Dataset

We are given a dataset on the historical stock price for Nvidia for the last 5 years. The dataset has 1258 entries starting 10/27/2016 to 10/26/2021. The features include information such as the price at which the stock opened, the intraday high and low, the price at which the stock closed, the adjusted closing price and the volume of shares traded for the day.

Below is the representation of some data from NVDA historical stock price dataset:

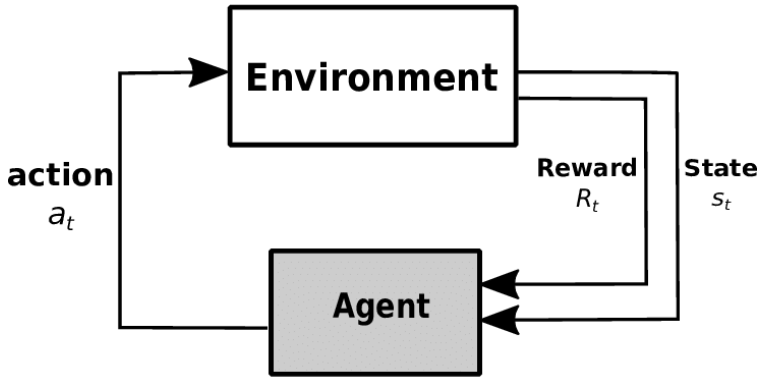
Date	Open	High	Low	Close	Adj Close	Volume
27-10-2016	18.1775	18.2125	17.5975	17.67	17.416187	38866400
28-10-2016	17.754999	18.025	17.6075	17.639999	17.386621	29085600
31-10-2016	17.6975	17.907499	17.6875	17.790001	17.534472	25238800
01-11-2016	17.855	17.952499	17.0725	17.262501	17.014545	47322400
02-11-2016	17.395	17.629999	17.16	17.190001	16.943085	29584800
03-11-2016	17.27	17.285	16.66	16.99	16.745956	30966400
04-11-2016	16.877501	17.182501	16.645	16.8925	16.649853	32878000
07-11-2016	17.387501	17.93	17.375	17.817499	17.561563	48758000
08-11-2016	17.885	17.942499	17.625	17.790001	17.534472	42988400
09-11-2016	17.307501	17.725	17.18	17.49	17.238771	45653200
10-11-2016	17.872499	17.875	16.690001	16.942499	16.699137	86928000
11-11-2016	19.877501	22.192499	19.625	21.9925	21.676601	217534400

3. Python Editor

I have used Jupiter Notebook on Google Collab for implementation and shared.

3.1 Environment

In Reinforcement learning, Environment is the place which allows Agent to observe State. When the Reinforcement Learning agent performs the action, the environment will respond to that action and transitions to next state along with providing reward or penalty for that action.



The environment has below methods which are already provided.

Init method: This method initializes the environment and takes the input parameters as below.

Input parameters:

1. file_path: Path of the CSV file containing the historical stock data.

2. train: - Boolean indicating whether the goal is to train or test the performance of the agent.

3. number_of_days_to_consider :- Integer representing whether the number of days the for which the agent considers the trend in stock price to make a decision.

Reset method: This method will reset the environment and returns the observation.

- This method will return an observation (Integer) in the range of 0 to 3 representing the four possible observations that the agent can receive. The observation depends upon whether the price increased on average in the number of days the agent considers, and whether the agent already has the stock or not.

Step method: This methods will let us know what will happen when the agent performs the actions (Buy/Sell/Hold)

- This method takes Integer (0 for Buy / 1 for Sell / 2 for Hold) as Input and returns 4 parameters (observation, reward, done and info)

- observation: This is an integer value of range 0 to 3 which represents four possible representations that the agent can receive. This observation depends upon whether the price increased on average in the number of days the agent considers, and whether the agent already has the stock or not.
- reward: This is an Integer/Float value which is specify the performance of the agent.
- done: This is a Boolean value which specifies whether an episode has ended or not.
- info: This is a dictionary which can be used to provide additional implementation information.

Render method: This method will render the agent's total account value over time. This takes an input parameter (mode).

4. Implementing Q-learning

Q-learning algorithm

- The goal of the agent is to increase the total rewards it will obtain from the environment. This function to maximize is known as discounted return function which is given as G.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \end{aligned}$$

- For maximizing the rewards, the agent needs to find an optimal policy and this Optimal policy is given by Bellman Optimality Equation.

$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)}^{\text{learned value}}$$

Here Q is **Action-Value** function or **Q-value** function

- α (Alpha) in the equation is the learning rate which controls convergence. It also determines how our Q-values are updated.
- γ (gamma) Is the discount factor which determines how much importance we have to give to future rewards. A high value for the discount factor will capture long-term rewards.
- So, when the agent performs the action, the Q-value of the agent's current state and action is stored in a **Q-table.**
- Q-table is a matrix where we have row for each State and column for each action.

Exploration-Exploitation Trade-off

- The agent has no idea about the environment in the beginning, So the agent will need to explore the environment rather than to exploit straight away.
- So there's a need for trade-off between exploration (choosing a random value) and exploitation (choosing actions based on Q-values).
- Also, we need to prevent overfitting i.e preventing the agent to take the same route always, so we introduce another parameter Epsilon (ϵ) to handle this.
- Instead of always selecting the best values from Q-table, we will sometimes explore the action space.
- This can be achieved by decaying the epsilon value every episode using exponential decay formula.

$$P = P_0 e^{-kt}$$

Steps to implement Q-learning algorithm

Initialization:

- Initially we declare the Q-table with all zeros.

```
q_table = np.zeros((q_table_state_size, q_table_action_size))
print(q_table)
print("q-table shape: ", q_table.shape)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
q-table shape:  (4, 3)
```

- Now we initialize the parameters for Exploration and Hyper-parameters for training and Lists to store accumulated rewards, episodes and epsilon decay.

```
# Parameters for Exploration
self.epsilon = 1.0                      # Exploration rate parameter
self.maximum_epsilon = 1.0              # Exploration probability which is at Maximum at start
self.minimum_epsilon = 0.01             # Minimum exploration probability
self.rate_of_decay = 0.005              # Rate of Decay for Exploration

# Hyperparameters
self.episodes_total = 2000              # Total number of episodes
self.l_r = 0.05                         # Learning rate
self.gamma = 0.90                       # Discounting rate

# Lists to store totals
self.rewards = []                       # List of rewards
self.episodes = []                      # List of Episodes
self.total_rewards = 0                  # Total Rewards
self.epsilon_decay_total = []           # Total Epsilon decay
```

Training:

- Now, we train the agent by picking a random number between 0,1 and check with exploration-exploitation trade-off parameter.

```
exploration_exploitation_rand = np.random.uniform(0, 1)
```

- If this parameter is greater than Epsilon, we perform Exploitation, else we perform Exploration.

```
action = np.argmax(q_table[state,:]) if exploration_exploitation_rand > self.epsilon else self.environment.action_space.sample()
```

- This action is passed to the step method which returns us the new state, reward.
- We then select the highest Q-value from the Q-table for the state (S').
- Update the Q-table using the Bellman Optimality Equation.

```
q_table[state, action] = (1 - self.l_r) * q_table[state, action] + self.l_r * (reward + self.gamma * np.max(q_table[new_state, :]) - q_table[state, action])
```

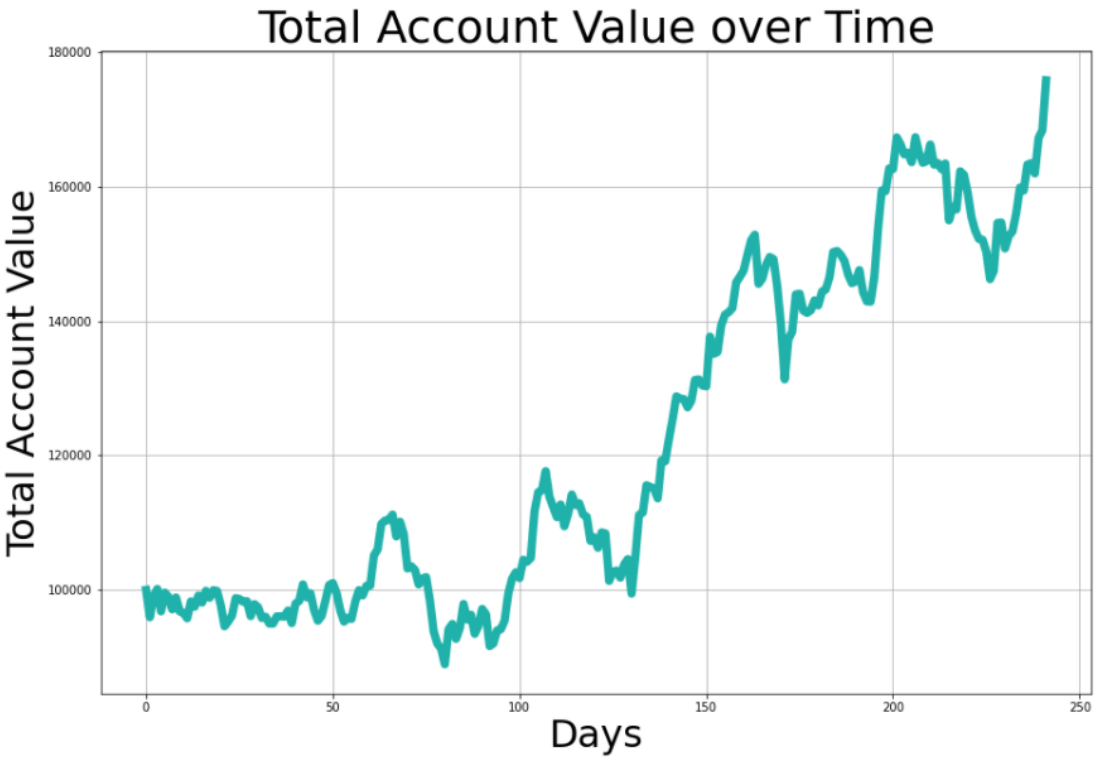
- Using the result of the previous action, we navigate to the next state (S').
- If the episode is done, then end the step and repeat for number of iterations.
- We then store the total rewards, episodes and the decaying epsilon values to new lists.
- Q-table and Total Account Value is obtained as below after training.

```
q-table after training :
 [[ 4.15777076 -3.13242718  0.67167151]
 [-0.67712549  4.22394987 12.12979895]
 [ 2.36212888 -3.92109784  0.64385297]
 [-3.22718472  2.5614617  -2.6421043 ]]

Total Account value for train:  844020.3820830005
```

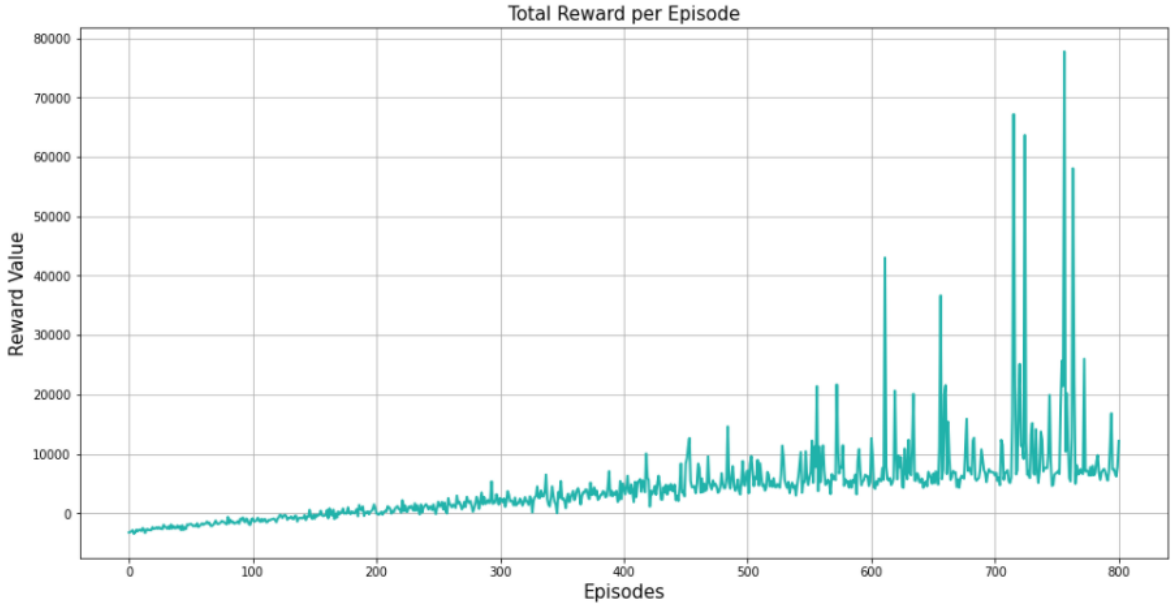
Evaluation:

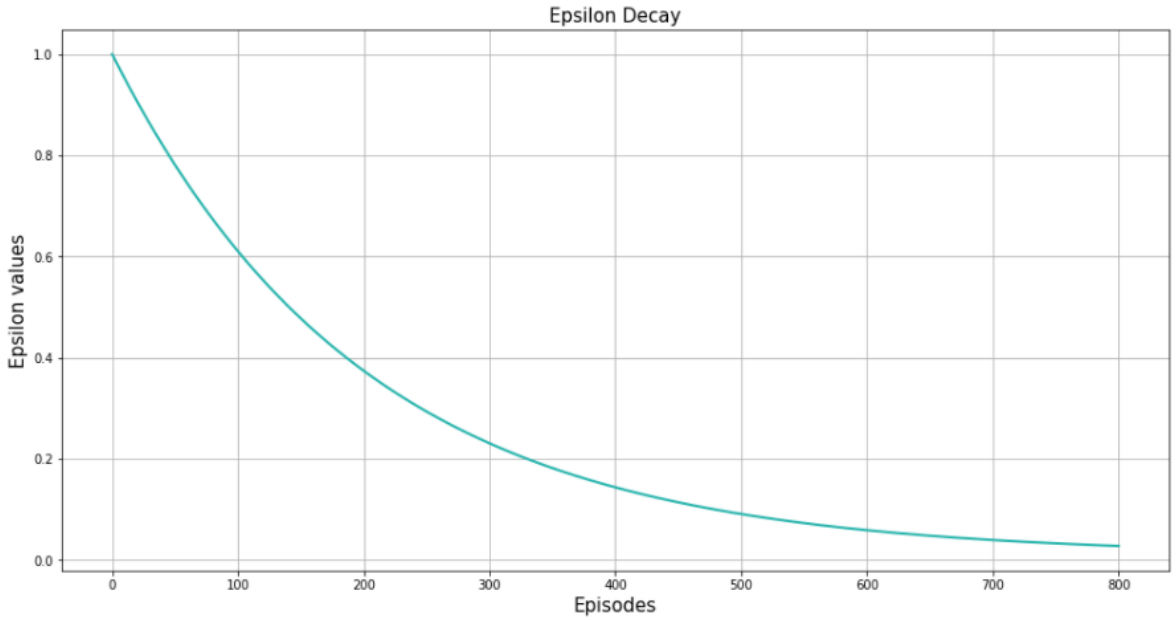
- As a first step, we reset the environment and rewards.
- Now we take the max value from Q-table generated from training and perform only exploitation.
- This action is passed to the step method and the new_state, reward, done and info are achieved.
- With these values, we update our Q-table further and store the total rewards.
- Once the evaluation is done, we print the Q-table and Total Account Value over time.

```
q-table for evaluate :
 [[  4.54572317  -3.13242718   0.67167151]
 [ -0.67712549   4.22394987 185.97489542]
 [  2.04644604  -3.92109784   0.64385297]
 [ -3.22718472  -5.3169907   87.81703222]]

Total Account value after evaluating :  175852.10231000002
```



➔ We now plot the graphs for **Total Reward per Episode** and **Epsilon Decay**





References:

1. Understanding the Bellman Optimality Equation in Reinforcement Learning (https://www.analyticsvidhya.com/blog/2021/02/understanding-the-bellman-optimality-equation-in-reinforcement-learning/)
2. Reinforcement Learning in Trading (https://blog.quantinsti.com/reinforcement-learning-trading/)
3. Q-learning in Reinforcement Learning (https://www.youtube.com/watch?v=DhdUlDIAG7Y)