PROJECT REPORT

Q-LEARNING ON STOCK TRADING ENVIRONMENT

SOUMIL GHOSH

UB ID - 50400736

EMAIL –
SOUMILGH@BUFFALO.EDU

Introduction

Our task in this project was to perform Q-Learning on the Stock Trading Environment. The Stock Trading Environment is based on the GYM library environment. Our goal in this task was to find a trading strategy with which we can maximize our investment capital or account value by taking proper actions based on the current state of the market.

Dataset

In this project we have worked on the NVIDIA dataset which contains the historical record of the stock price of NVIDIA starting 10/27/2016 to 10/27/2021. The dataset has the following parameters:

- Price at which stock opened
- Intraday high Stock Price
- Intraday Low Stock Price
- Price at which stock closed
- Adjusted closing Price
- Volume of shares traded for the day

Stock Trading Environment

The Stock Trading Environment has been designed to train the agent to take proper decisions on whether to Buy, Hold or Sell the stock based on the current situation of the market. We are starting with a total account value of $100,000 and our goal is to maximize this account value to guarantee maximum return on our investment.

Methods:

Init method: This method initializes the environment.

Input parameters:

- file_path: -Path of the CSV file containing the historical stock data.
- train: - Boolean indicating whether the goal is to train or test the performance of the agent.
- 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 resets the environment and returns the observation.

Returns: 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 method implements what happens when the agent takes the action to Buy/Sell/Hold.

Input parameter: action: - Integer in the range 0 to 2 inclusive.

Returns:

- 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.
- reward: - Integer/Float value that's used to measure the performance of the agent.
- done: - Boolean describing whether the episode has ended.
- info: - A dictionary that can be used to provide additional implementation information.

Render method: This method renders the agent's total account value over time.

Input parameter:

mode: 'human' renders to the current display or terminal and returns nothing.

Q-Learning Algorithm

Initialization of Parameters

In the first step of the Q-Learning algorithm, we are initializing key parameters like number of episodes, a 4*3 Q-table, discount factor, learning rate, epsilon and two lists, one for keeping a record of cumulative reward in each episode and one for recording the epsilon decay with time.

Training

We train our agent for 10 episodes. In each episode, we are considering 100 days during which our agent must take an appropriate decision to Buy, Hold or Sell the stock based on the market situation. In each training episode we are performing the following steps:

1. First, we reset the environment by setting the environment.train variable to true
2. We select a random value from 0 to 1 and compare it with our epsilon value. If our epsilon value is greater, we choose to explore else we choose to exploit by selecting the action with the maximum Q [State, Action] value and call the environment.step() function using this action
3. Then we update the Q-table using the Bellman's Equation also increment our total rewards
4. Lastly, we updated our epsilon value for each step
5. We continue steps 2 to 4 until our done value becomes TRUE
6. Finally, we append the total rewards of the particular episode to our cumulative_reward list

We have used two methods for training our agent:

- Linear Epsilon Decay - In this scenario, we reduced our epsilon value in a linear fashion
- Quadratic Epsilon Decay – In this scenario, we reduced our epsilon value in a quadratic fashion

Evaluation

The evaluation of the agent is almost same as the training with a few differences. The following are steps for evaluation:

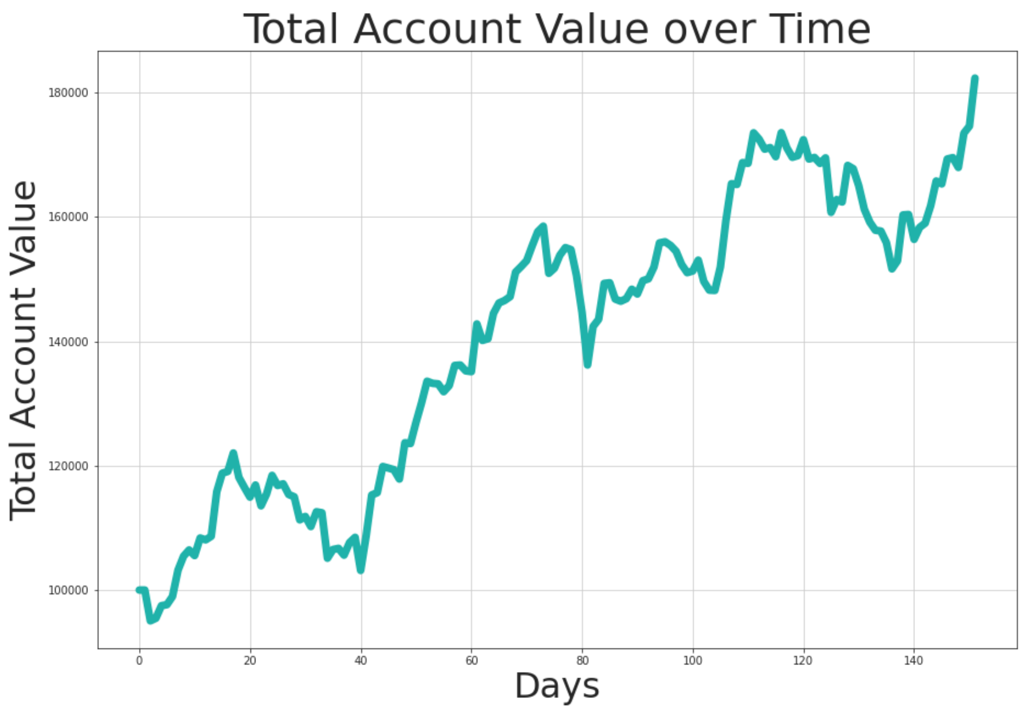
1. First, we turn our environment.train variable to False and reset the environment
2. Next, we select our next action as the maximum value of the Q-table for that state and call environment.step() function and increment our total_reward based on the output we get
3. We continue steps 1-2 until our done value becomes TRUE

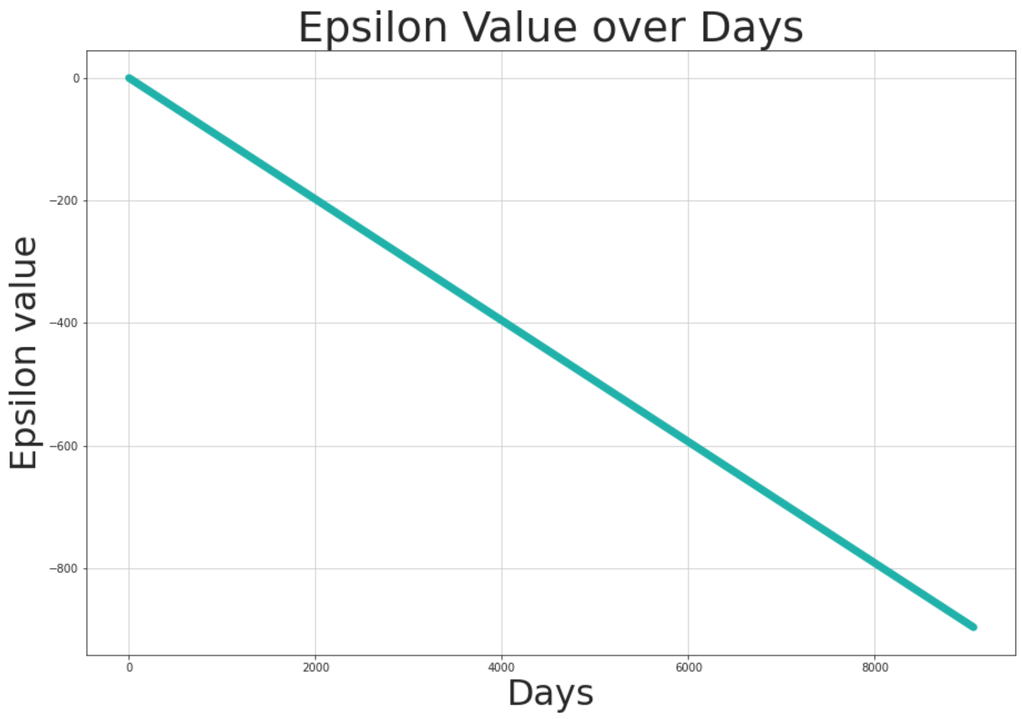
Results

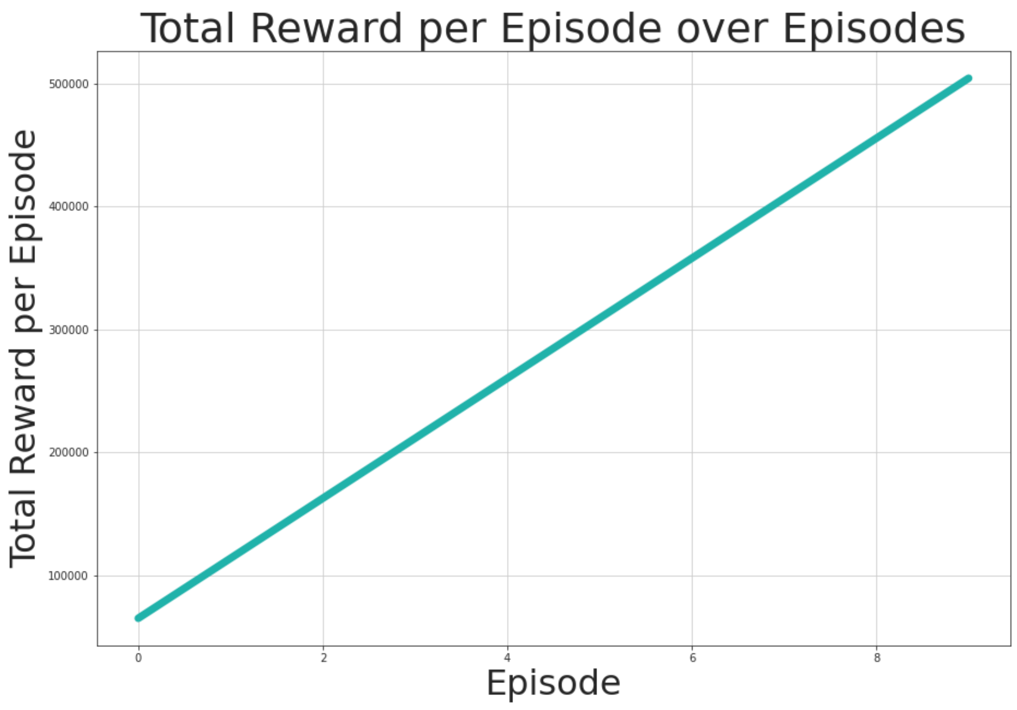
Case -1 – Linear Epsilon Decay

After evaluating our agent, we achieved a total account value of $182,366.90 which is a 82.3669% return on our investment capital of $100,000.

We have plotted the Total Account Value over Time, Epsilon Decay over time and Total Reward over Episodes for a better understanding of the results.



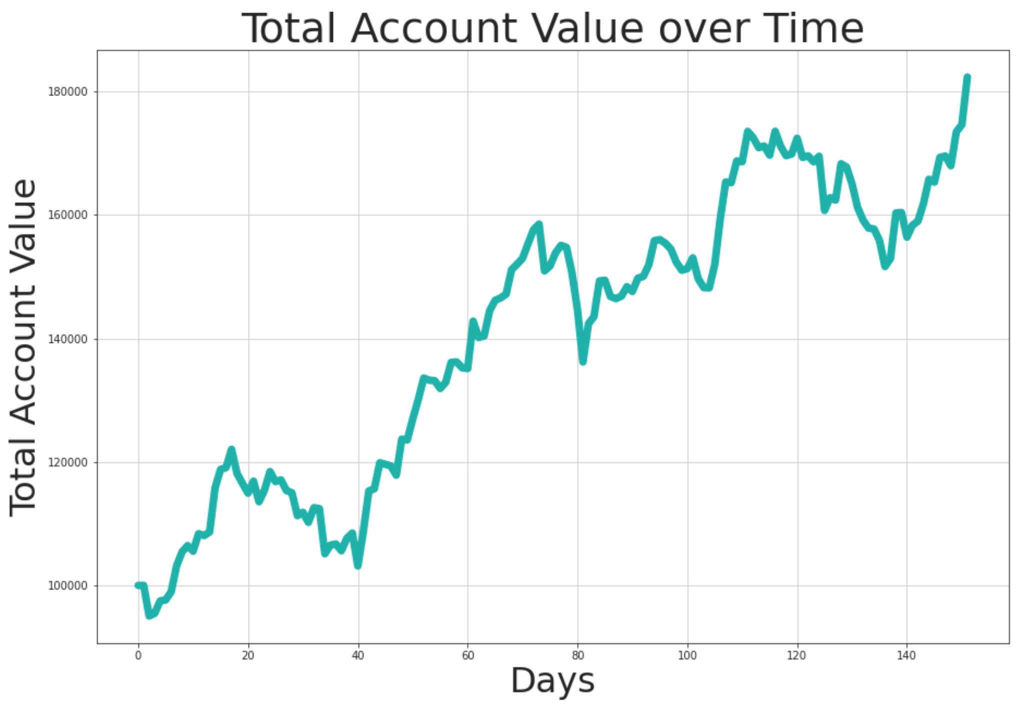

Epsilon Value over Days
Epsilon value
Days
0
−200
−400
−600
−800
0
2000
4000
6000
8000

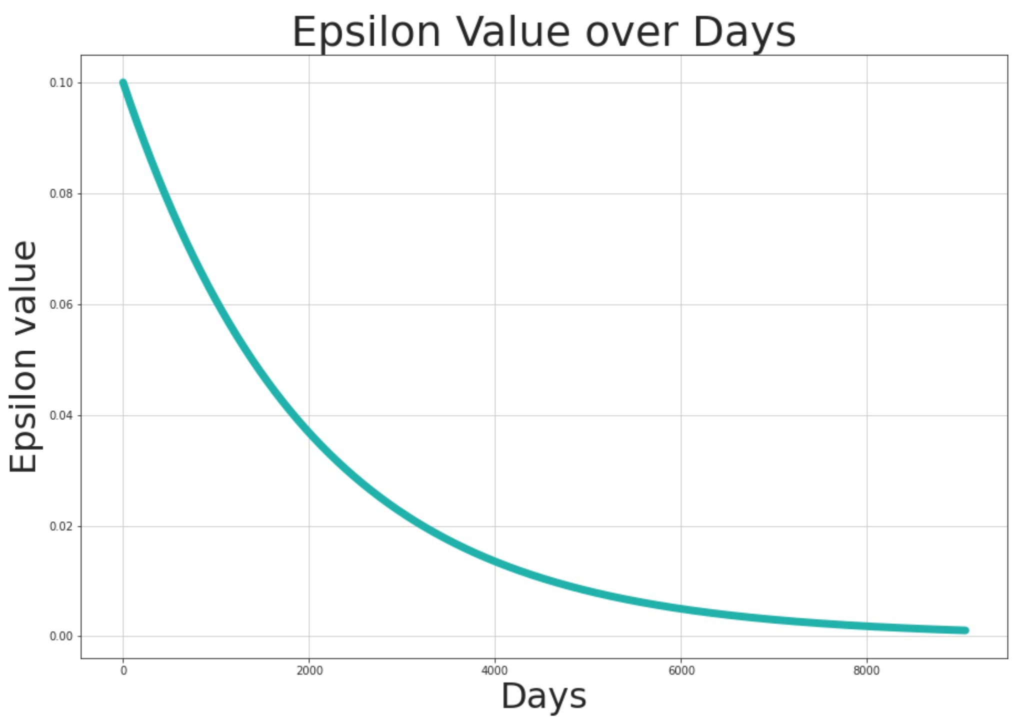

Total Reward per Episode over Episodes
Total Reward per Episode
Episode
500000
400000
300000
200000
100000
0
2
4
6
8

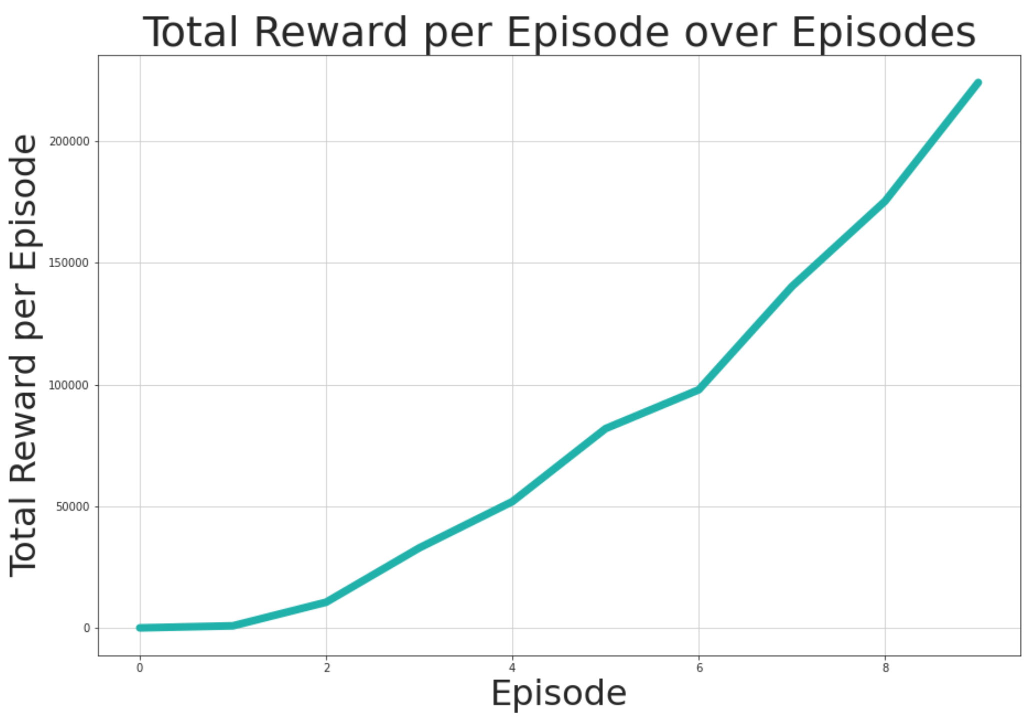
Case – 2-Quadratic Epsilon Decay

After evaluating our agent, we achieved a total account value of $182,366.90 which is a 82.3669% return on our investment capital of $100,000.

We have plotted the Total Account Value over Time, Epsilon Decay over time and Total Reward over Episodes for a better understanding of the results.







The point to be noted over here is that we are getting similar return of investment in case of both Linear and Quadratic Epsilon Decay after proper optimization. There are some differences in the intermediate stages in case of Total Account Value over Time and Total Reward per episode as is evident from the graphs but in the end, we find that the return of investment, we get in both cases is similar.

Conclusion

Through this project we have learned how to implement Q-Learning on an environment. We also learned how the agent can gradually optimize its performance based on the feedback it receives from the environment.

Another crucial finding in this project was that the return of investment we get in case of both linear and quadratic epsilon decay are very identical, after proper optimization. So, from here we can conclude that the performance of the agent is not always dependent on whether we use linear or quadratic epsilon decay while training.