Assignment 3: - Deep Reinforcement Learning

Rishi Jay Joshi
Department of Computer Science and Engineering
University at Buffalo, SUNY
New York, United States
rishijay@buffalo.edu

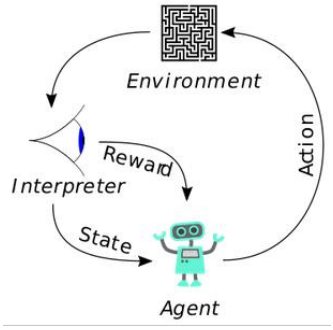
Abstract

Assignment 3 deals with implementing the Q-Learning algorithm in python. The goal of the assignment 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 the agent can either buy, sell or hold. We start with an investment capital of $100,000 and the agent's performance is measured as a percentage of the return on investment. Q-Learning is used to train an agent to learn the trends in stock price and perform a series of trades. At the end of this assignment, we will understand the benefits of using reinforcement learning to solve the real world problem of stock trading.

1 Introduction

1.1 Reinforcement Learning

[1] Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. A typical format of reinforcement learning scenario is shown below



1.2 Q-Learning

[2] Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the

environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations.

2 Experiment

2.1 Dataset

We have been 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.

2.2 Environment Structure

We have been given an environment which calculates the trends in the stock price. We will use the Q-Learning algorithm on this environment to figure out a trading strategy and increase our total account value over time.

Init method: This method initializes the environment.

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 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:

1. 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.
2. reward: - Integer/Float value that's used to measure the performance of the agent.
3. done: - Boolean describing whether or not the episode has ended.
4. 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

2.3 Python Editor

Google Colab based on jupyter notebook is used for implementation

2.4 Implementing Q-Learning

[3] Learning the Q function corresponds to learning the optimal policy. The key problem is finding a reliable way to estimate training values for Q given only a sequence of immediate rewards r spread out over time. This can be accomplished through iterative approximation.

[2] The algorithm, therefore, has a function that calculates the quality of a state–action combination.

$$Q : S \times A \to \mathbb{R}.$$

Before learning begins, Q table is initialized to zeroes. Then, at each time t the agent selects an action, observes a reward, enters a new state and Q is updated. The core of the algorithm is a Bellman equation as a simple value iteration update, using the weighted average of the old value and the new information:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

[4] Epsilon Greedy is one of the most popular strategies to balance exploration and exploitation.

- ε is assigned a real number between 0 and 1.
- With probability ε the agent chooses a random action.
- With probability 1-ε the agent acts greedily.

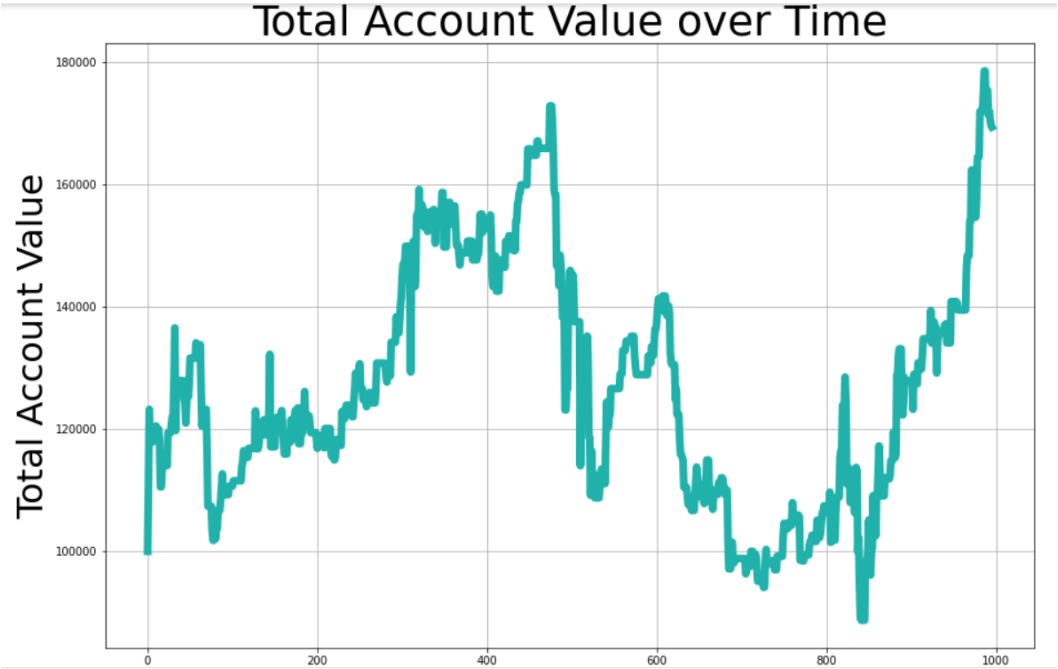
In reward based decay only when an agent has crossed some reward threshold, the value of ε is reduced. Instead of assuming that the agent is learning more every episode, we wait for proof of agent's learning before reducing the epsilon value. Thus we set higher targets for agent every time ε is lowered, and wait for agent to reach the newer target before repeating the same steps again.

```
if epsilon > min_epsilon and last_reward >= reward_threshold:
    epsilon = decay_epsilon(epsilon)
    reward_threshold = increment_threshold(reward_threshold)
```

3 Results

After successfully implementing the Q-Learning algorithm, we get the following results.

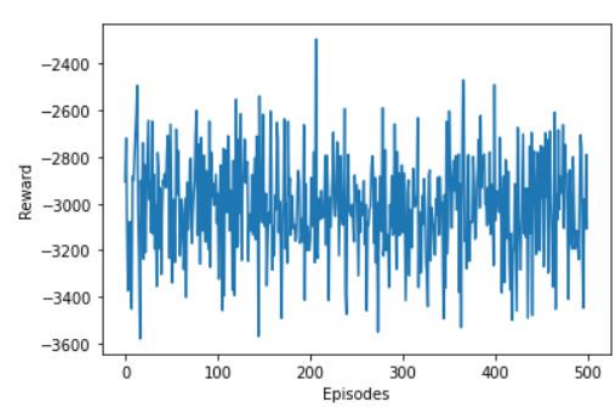
The total account value over time is as follows:-

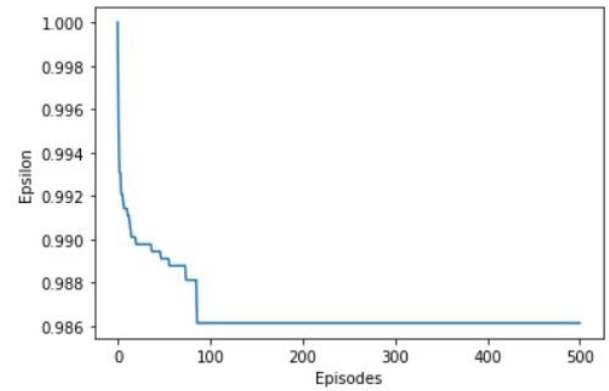


After training we get the following total account value

```
169461.1920370001
```

We get the following plots after training

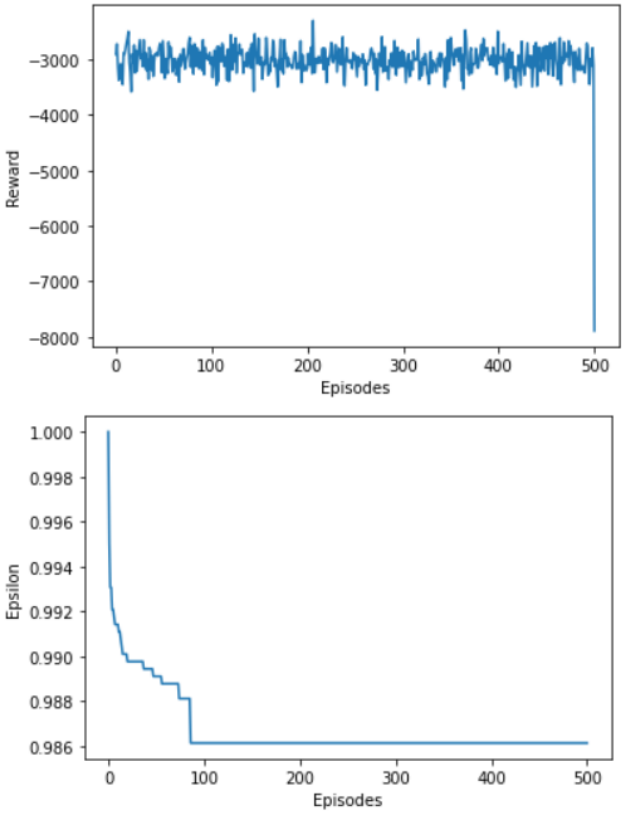




After evaluating we get the following total account value

```
134859.52313799987
```

We get the following plots after evaluating



4 References

[1] https://en.wikipedia.org/wiki/Reinforcement_learning

[2] https://en.wikipedia.org/wiki/Q-learning

[3] Class Slide – Q-Learning

[4] https://aakash94.github.io/Reward-Based-Epsilon-Decay/