Inventory Management

Lidor Erez

June 2024

1 Preface

This is my attempt to use the knowledge I acquired on reinforcement learning so far to implement from scratch the inventory management problem defined in Stanford CS234_A1.

2 Introduction

The inventory management problem is a classic example of a decision-making task where an agent must optimize its actions to achieve the highest possible reward. In this scenario, the agent is responsible for managing the inventory levels of a store. The problem is modeled as a Markov decision process (MDP), where the state represents the current stock level, and the agent can choose between two actions: buy or sell. The objective of this report is to implement the inventory management problem from scratch using Python, leveraging the concepts of reinforcement learning acquired so far. The implementation will involve defining the reward function, modeling the MDP, and evaluating the agent's performance under various conditions. Furthermore, I will analyze how the agent's decision-making process evolves as a function of the discount factor γ, which influences the agent's consideration of future rewards.

3 Background

In this section I will define the problem using mathematical notation and detail my implementation.

3.1 The Problem

Let s be the stock level of the inventory at time step t where $s \in [0, 10]$. At any time, the agent has to decide between two actions buy or sell which yields different rewards given the state it is in. This can be defined as follows:

- $R(s, sell, s-1) = 1$ if s > 0

- $R(s, sell, s-1) = 0$ if s $= 0$
- $R(s, buy, s+1) = 0$ if s < 9
- $R(s, buy, s+1) = 100$ if s $= 9$

We assume that the initial state is $s = 3$ at the start of every episode. In the task, the number of time steps is 4; however, in my code, each episode is of a length of 20.

3.1.1 Reward Function Implementation

The reward function $R(s, a)$ is defined as:

$$R(s,a) = \begin{cases} 1, & \text{if } a = \text{sell and } s > 0 \\ 0, & \text{if } a = \text{sell and } s = 0 \\ 0, & \text{if } a = \text{buy and } s < 9 \\ 100, & \text{if } a = \text{buy and } s = 9 \\ 0, & \text{otherwise} \end{cases}$$

And the next state s' is given by:

$$s' = \begin{cases} s-1, & \text{if } a = \text{sell} \\ s+1, & \text{if } a = \text{buy and } s < 9 \\ 10, & \text{if } a = \text{buy and } s = 9 \\ s, & \text{otherwise} \end{cases}$$

3.1.2 Action-Value Function Implementation

The Action-Value function $Q(s, a, \gamma, H, \text{stop})$ computes the expected reward for taking action a in state s, considering the discount factor γ and the horizon H. It uses the Bellman equation to recursively calculate the optimal policy:

$$Q(s,a,\gamma,H,\text{stop}) = \begin{cases} 0, & \text{if } H \geq \text{stop} \\ R(s,a), & \text{if } \gamma = 0 \\ R(s,a) + \gamma \cdot \max(Q(s', \text{sell}, \gamma, H+1, \text{stop}), Q(s', \text{buy}, \gamma, H+1, \text{stop})), & \text{otherwise} \end{cases}$$

Where H is the current time step, and *stop* is the maximum time step for each episode. The function $Q(s, a, \gamma, H, \text{stop})$ evaluates both possible actions ('buy' and 'sell') from the next state and chooses the one that maximizes the expected reward. This way, the Bellman equation is utilized to find the optimal policy $\pi^*(s)$ recursively..

3.1.3 Episode Simulation

The episode simulation function models a single run of the inventory management task, where the agent interacts with the environment to maximize its rewards over a fixed number of time steps. Here's how it works step-by-step:

Pseudo Code:

```
function episode(H, s, gamma, stop):
    initialize rewards as an empty list

    while H <= stop:
        q_buy <- Q(s, 'buy', gamma, H, stop)
        q_sell <- Q(s, 'sell', gamma, H, stop)

        if q_buy > q_sell:
            action <- 'buy'
        else:
            action <- 'sell'

        reward, next_state <- R(s, action)
        append reward to rewards list

        s <- next_state
        H <- H + 1

    return rewards
```

This function ran with 3 different values of $\gamma \in \{0, 0.3, 0.7\}$.

4 Results & Conclusions

As mentioned above, my part of my goal was to understand the influence of γ on the cumulative reward and agent's decisions. The results shows that as γ gets bigger, the agent decision is mainly buying rather than selling as buying eventually end up with higher cumulative reward. On the other hand as γ approaches to 0 the agent care more about immediate rewards rather than rewards in the far future as illustrated in figure 1. Furthermore, it seems that when $\gamma = 0.7$ the agent learns that restocking and then selling $\rightarrow$ buying leads to the most reward over time. Namely, this agent spend the first 7 time steps to restock the inventory and then sell and buy to get the maximum reward as illustrated in 2.

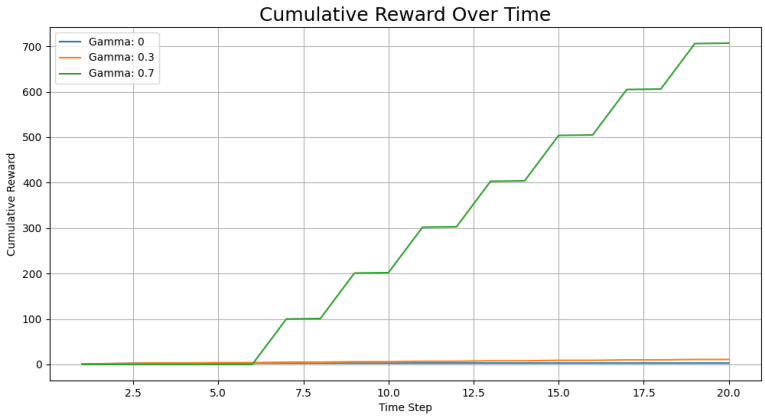

Figure 1: Cumulative reward over time for $\gamma \in \{0, 0.3, 0.7\}$

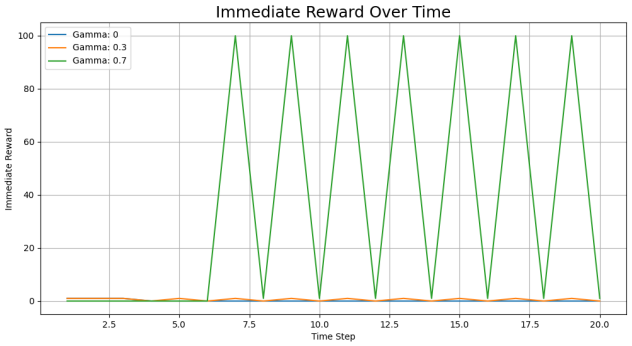

Figure 2: Immediate rewards for $\gamma \in \{0, 0.3, 0.7\}$

5 Appendix

For the full Python code: Inventory Problem

References

[1] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[2] Stanford University. (2019). *Reinforcement Learning Course Winter 2019* [YouTube playlist]. YouTube. Retrieved from `https://www.youtube.com/playlist?list=PLoROMvodv4rOSOPzutgyCTapiGlY2Nd8u`