AI for Multi-period Inventory Optimization with Uncertain Demand

Project report submitted in partial fulfillment of the requirements for the degree of

Masters of Business Administration

By

Pratik Kumar Mitra

20810082

MBA Batch 2020-2022

Under the guidance of

Dr. Manu K. Gupta

Assistant Professor

DoMS, IIT Roorkee

Department of Management Studies

Indian Institute of Technology

30th April 2022

Contents

Acknowledgement

It gives me immense pleasure to express my gratitude to many individuals for their cordial cooperation and encouragement, who have contributed directly or indirectly in successfully completing my final project and preparing this report. First, I would like to express my gratitude to my project mentor, Dr. Manu K Gupta, for his continuous guidance and unyielding support throughout the duration of the project.

I would also like to thank Dr. Usha Lenka, Head of the Department, Department of Management Studies, IIT Roorkee, and all other faculty members who provided me with the facilities required and conducive conditions for completing this project.

Thank You.
Pratik Kumar Mitra
Enrollment No.-20810082

1 Abstract

An inventory management problem is addressed at a particular node in the supply chain which experiences uncertain demand across multiple periods. This study is carried out for a single product. A Reinforcement Learning based dynamic algorithm is proposed to tackle the problem. It is tested on an uncertain but known probability distribution of demand and then on actual demand data. The performance is then compared against three basic inventory replenishment policies 1) Restocking when no inventory is left; 2) Restocking at every opportunity and 3) Restocking when inventory is below a certain threshold. The study is carried out in a simulated environment created using open source libraries in Python. The results are as expected, where the Q-Learning algorithm performs better as compared to the standard restocking policies. A better performance is achieved when considering the Quality of Service paradigm within our algorithm.

Keywords - Reinforcement Learning; Inventory Management; Markov Decision Chain; Q-Learning; Quality of Service.

2 Introduction

Inventory is the stock of products that every level of the supply chain needs to maintain to ensure orders are readily fulfilled. Retailers need to balance the inventory costs between ordering costs and holding costs on the one hand and lost revenue and stock-out expenses on the other. At the elementary level, with deterministic and stable demand patterns, we try to optimize inventory using the Economic Order Quantity formula as given below:

$$EOQ = \sqrt{(2 * D * S)/H} \quad (1)$$

where D is the deterministic annual demand, S is the setup cost for each order, and H is the holding cost per unit.

This basic case considers that the annual demand is distributed equally throughout the year, which is generally not the case in practical situations. When evaluating lead times, retailers need to hold a certain amount of safety stock to keep selling during the lead time so that they do not incur stock-out costs. The fundamental formula for calculating the safety stock is:

Safety Stock = Average lead-time * average demand

The above equation works only when the lead times and demands within the supply chain are deterministic. It is also not equipped to handle sudden changes and the bull-whip effects that the present-day supply chains face from global factors. Hence, the motivation is to develop an agent that can learn from the available data and help optimize inventory decisions across multiple periods, considering any real-life supply chain's practical issues.

2.1 Objectives of the project

- Create a robust, Q-Learning agent to learn the inventory decisions for uncertain demands across multiple periods.
- Compare the agent's performance initially for known demand distributions against the optimum policy generated using the Value Iteration method.
- Compare the agent's performance for actual demand values against standard inventory replenishment strategies.
- Test for scalability and robustness.

3 Literature Review

The paper by Meisheri, Hardik, et al. [1] addresses a critical challenge in a supply chain system for optimizing inventory replenishment, i.e., managing uncertainty in lead times and real-time information sharing across several echelons. This method takes advantage of the delay-resolved paradigm in the Reinforcement Learning literature to account for stochasticity at all levels computationally efficient, with lead times seen as action delays within the supply chain system. In contrast to the present RL-based inventory management system, the framework requires one agent to be trained for various lead time values. The model simply adds prior stocking decisions to its knowledge state and, in this way, eliminates the need to work with predicted demand or policy roll-outs. As a result, the model may be quickly trained and scaled to account for cross-product limitations. This paper is the first such study of its kind that simultaneously considers all aspects of supply chain inventory control in a computationally efficient way.

The beer game is a multi-agent, decentralized, collaborative supply chain problem. According to the paper by Oroojlooyjadid, Afshin, et al. [2], for specific instances, a base-stock inventory policy represents the optimal policy, but when some of the agents do not follow a base-stock policy (as is typical in real-world supply chains), the remaining players' best policy is unknown. The paper presents a DQN-based approach to overcome this problem. As the paper proposes a nonlinear approximator, there is no guarantee that it will result in the global best solution, yet it works well in reality. Playing the game alongside agents who follow a base-stock policy achieves near-optimal results. It outperforms a base-stock policy when the other agents employ a more realistic model of ordering behaviour. The model's training may be sluggish, but its execution is lightning fast. Furthermore, the technique uses just historic data and does not require knowledge of the demand probability distribution.

A transfer-learning strategy is proposed to minimize the calculation time necessary to train new agents with varying cost coefficients or action spaces. This method requires less time to train new agents since it eliminates the need to tune hyperparameters and has fewer trainable variables. It's also fairly robust, resulting in beer game costs comparable to fully trained agents while cutting training time by order of magnitude.

Another paper by Gijsbrechts, Joren, et al. [3] states that deep reinforcement learning can effectively tackle classic, intractable inventory issues. The A3C algorithm may be trained to generate policies comparable to those generated by state-of-the-art heuristics and other approximate dynamic programming approaches. Contrastingly, deep reinforcement learning is not simple to implement. Initial hyper-parameter tweaking is computationally and time-consuming, requiring both art and science. Nonetheless, in the other problem settings, after intensive manual adjustment of nine hyperparameters in one issue setting, fixing six parameters, and automatically modifying the learning rate, the entropy regularisation and buffer size resulted in a good performance. All sensitivity analyses performed well when only one well-performing hyper-parameter set was used. DRL appears to be a viable general-purpose solution with minor alterations across three diverse problem contexts. The controlled lab environment for which the stochastic inventory policies were designed was essential for the rigorous comparative study. The A3C method effectively solved the problem with the actual data sets.

The paper by Perez, Hector D., et al. [4] is the closest paper we have found to the work we are planning to do. This paper uses the open-source package OR-Gym for general single-product, multi-period make-to-order supply networks with multiple production and inventory holding facilities. The work presents new solutions for handling inventory management difficulties (e.g., multi-stage stochastic programming and rolling horizon implementations for deterministic and stochastic models). In the setting of a four-echelon supply network with uncertain stationary demand, inventory management strategies generated using deterministic linear programming, stochastic linear programming, and reinforcement learning are compared and contrasted. The results suggest that the stochastic model produces better supply network profitability results.
On the other hand, the reinforcement learning model maintains the network in a way that could make it more resilient to network disruptions, indicating that AI could be useful in supply chain applications. Even though deterministic linear models neglect the supply network's stochastic nature, they solve in fractions of a second and produce results equivalent to the profitability of Reinforcement learning approaches. Studying the effects of non-stationary demand on the models utilized and minimizing the end-of-simulation impacts that have been described earlier are possible extensions to this work.

As part of this project, I look forward to understanding the concepts discussed in the papers mentioned above and applying them to optimize decisions in a General Inventory Control problem over multiple periods.

4 Project Methodology

Ensuring that we are able to handle dynamic changes in demand, we are modeling the Inventory Control system as a Markov Decision Process (MDP), as discussed in Puterman, 1994. Markov Decision Process is a special type of Sequential Decision Model.

A Sequential Decision Model is a system where at a specific period in time, a decision-maker or an agent selects a particular decision from among an available set of decisions, which results in an immediate reward (or immediate cost) for the decision-maker and the system moving to a new state from the preceding state based on a defined probability distribution. The actions available to the agent at any instance are based on the state the system is in at that

given time. The agent or the decision-maker will again face a similar decision problem in the following period. The only difference is that the system may have evolved to a new state. The agent wishes to take that action in each period, resulting in maximum reward or minimum cost in the long run. The pictorial representation is given below:

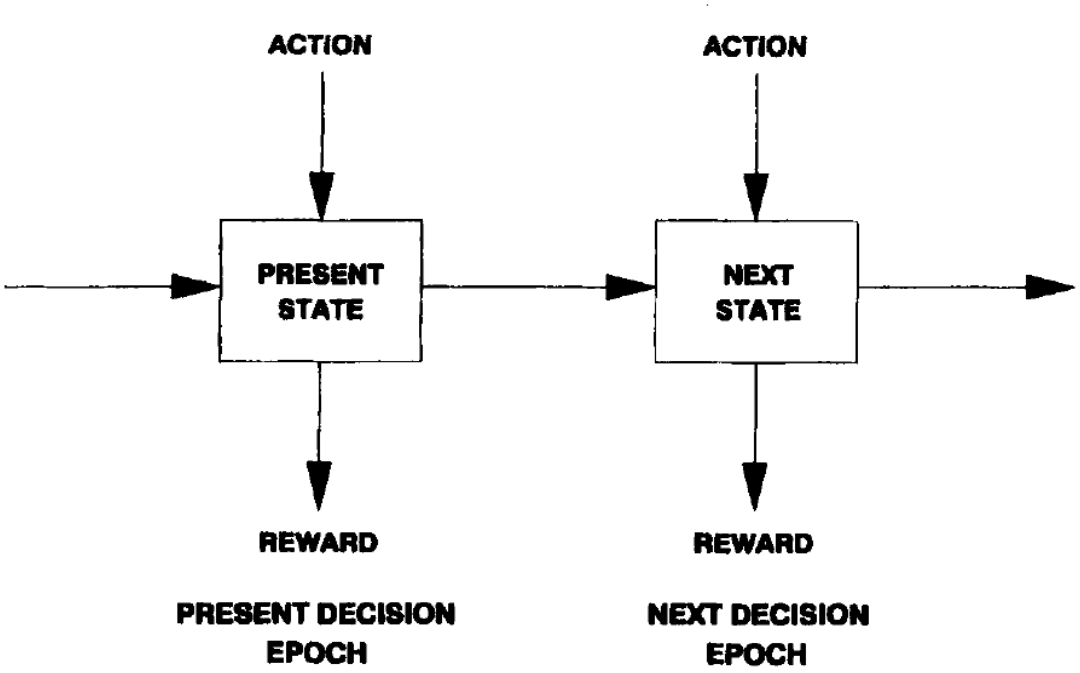

Figure 1– Symbolic representation of Sequential Decision Model – Puterman, 1994.

A Sequential Decision model is made up of the following key elements –

- A set of decision instances
- A set of system states
- A set of available actions
- A set of immediate rewards for each particular state-action combination
- A set of transition probabilities for each state-action combination

Markov Decision Process is a unique Sequential Decision Model. Here, the set of available actions, the rewards, and the transition probabilities depends only on the present state and action and not on the actions taken and states visited at previous decision instances to reach this state. Additionally, we get the Markov Decision Problem when an optimality criterion is added to the Markov Decision Process.

Symbolically, the Markov Decision Problem is represented as:

$$(S, A, T, R, \gamma)$$

where S represents the set of all available states, A represents the set of available actions, T represents the transition function, R is the reward function, and γ represents the discounting factor.

A policy for a Structured Decision Problem is defined as a unique selection of actions for each possible state given in the problem. It is a directive to the agent to choose a particular

action depending on the system's current state. Once a policy π is decided, we can calculate its value using the following formula:

$$\text{For s} \in \text{S}, V^{\pi}(s) = E_{\pi}[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots | s_0 = s\ ] \quad (2)$$

The above expression represents the value of the policy π for state s. When defined correctly, every MDP has to have an optimum policy π*. The optimum policy is defined as:

$$\forall \pi \in \Pi, \forall s \in \text{S}, \text{V}^{\pi*}(\text{s}) \geq V^{\pi}(s) \quad (3)$$

where Π is the set of all policies. There is a probability of having more than one optimal policy for an MDP, but the value function output for every optimal policy is the same, the unique value $V^{\pi^*}(s)$.

Another metric used to classify the actions to be taken for a given state is the Quality of Action Value function or simply the Q Values, which is defined as:

$$Q^{\pi}(s,a) = E[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots | s_0 = s, a_0 = a, a_t = \pi(s_t) for\ t \geq 1]$$

$$\text{or, } Q^{\pi}(s,a) = \sum_{s' \in S} T(s,a,s')\{R(s,a,s') + \gamma V^{\pi}(s')\} \text{ for } \pi \in \Pi \quad (4)$$

Hence, from the equation we just derived, it is clear that we need to know the transition and reward functions of an MDP to calculate the Q values. But, in a Reinforcement Learning setting, this is not the case. The agent only knows the states and available actions of the system. The agent needs to interact with the system to try to learn the optimum policy. It does so by being greedy in taking various actions and navigating the various states. During the exploration stage, the agent arbitrarily picks up an action depending upon the system's current state. During the exploitation stage, the agent only tries to take the optimum action for the current state, i.e., the action with the highest Q value.

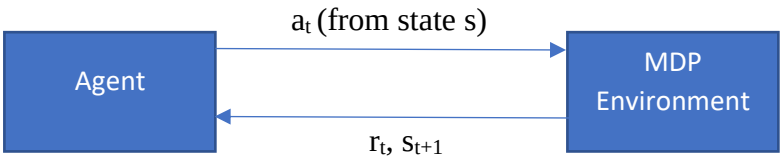

Figure 2– Schematic representation of a Reinforcement Learning Problem

Environment Definition – Initial testing of the RL algorithm was done in a randomized environment and compared against the optimum policy obtained using the value iteration process. The following random quantities were generated as part of the randomized environment:

- Lambda – The mean of the Poisson demand distribution
- M – The capacity of the warehouse or the maximum inventory allowed.
- K – The fixed cost of placing an order.
- alpha – The per unit ordering cost over and above the fixed cost.
- beta – The variable holding cost of each inventory unit per time period.
- gamma – The discounting factor used in the problem.
- delta – The revenue realized per unit of sales.

Modeling the problem - The problem is modeled for inventory decisions concerning a single product to meet uncertain demands across multiple periods and optimize costs simultaneously. The inventory manager needs to decide whether to order new stock from the supplier at every period. While doing so, he needs to balance the cost (ordering and holding cost for new inventory) and the lost sales and penalties, on the other hand. The demand is uncertain, and for the first case, we know the probability distribution of the demand.

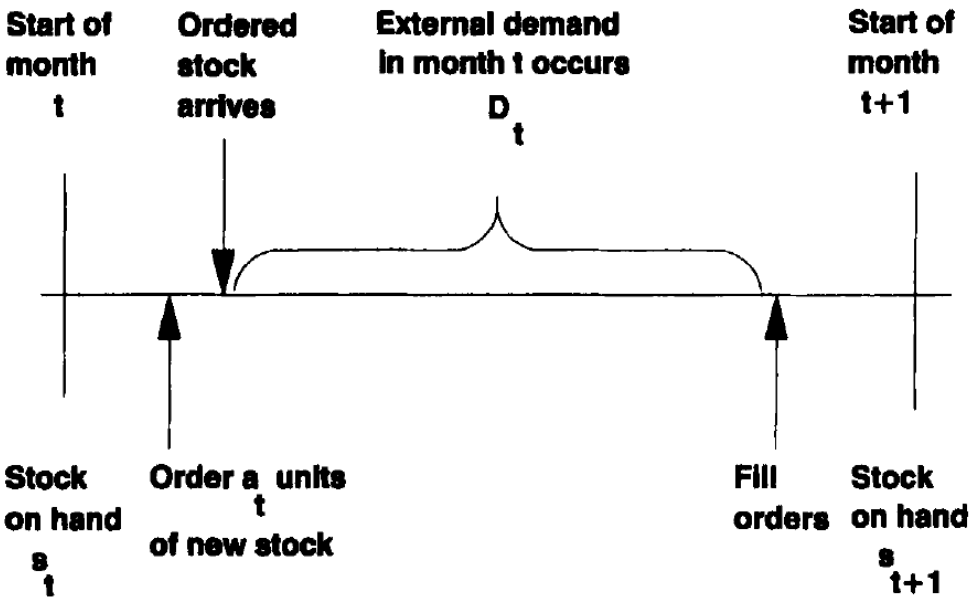

Figure 3– Representation of timing of events within a period – Puterman, 1994.

The following assumptions are made to help simplify building the model:

- The decision to order stock is made at the beginning of the month, and delivery occurs instantly.
- Demands are captured throughout the month, and deliveries are made on the last day of the month.
- No backlogging is considered, i.e., if demand exceeds inventory, the excess is lost.
- In the first case, the costs, revenues, and demand distribution do not vary from one month to another.
- The product is sold in whole units only.

Figure 3, shown above, introduces the following notations:

- s_t – Inventory on hand at the beginning of month t.
- a_t – Quantity if additional stock ordered at the beginning of month t.
- D_t – Random demand for the period t. The probability distribution of demand is known in the first case.
- s_{t+1} – Inventory on hand at the beginning of month t+1.

Since we are not backlogging orders, we get the below equation connecting the above quantities:

$$s_{t+1} = \max\{s_t + a_t - D_t, 0\} \equiv [s_t + a_t - D_t]^+ \quad (5)$$

The economic parameters are now defined below. All values are calculated as on the beginning of respective periods to implicitly capture the idea of the time value of money within our concept. The present value of ordering a_t units in any period is $O(a_t)$ and is composed of a fixed component K>0 and a variable component dependent on the ordering quantity.

$$O(a_t) = \begin{cases} K + alpha * a_t \; if \; a_t > 0 \\ 0 \qquad\qquad if a_t = 0 \end{cases} \tag{6}$$

The holding cost for any period t can be represented by $h(s_t + a_t)$ as:

$$h(s_t + a_t) = beta * (s_t + a_t) \tag{7}$$

The reward for the period t can be calculated by $r(s_t,a_t,s_{t+1})$ as:

$$r(s_t, a_t, s_{t+1}) = delta * (s_t + a_t - s_{t+1}) - O(a_t) - h(s_t + a_t) \tag{8}$$

4.1 Pseudo Code

The steps undertaken to arrive at the results for random demand are as given below:

- Step 1: Generate the environment values of Lambda, M, K, alpha, beta, gamma, and delta.
- Step 2: Since we know the probability distribution of demand, we calculate the transition probabilities. These are calculated using:

$$T(s,a,s') = \begin{cases} 0 & if \; M \geq s' > s + a \\ \dfrac{Lambda^{(s+a-s')}}{(s+a-s')!} e^{-Lambda} & if \; M \geq s + a \geq s' > 0 \\ \displaystyle\sum_{i=s+a}^{\infty} \frac{Lambda^i}{i!} e^{-i} & if \; M \geq s + a \; and \; s' = 0 \end{cases}$$

- Step 3: Next, we calculate the rewards for all possible combinations by using:

$$R(s,a,s') = r(s,a,s') \; \{From \; 8\}$$

- Step 4: Apply the Value Iteration method to find the optimum policy.
- Step 5: Design a Reinforcement Learning agent that takes in the current state, action, and Quality of Service Coefficient and returns the current reward, new state, and misses based on the demand generated using the Poisson distribution with mean as Lambda. Misses denoting the unfulfilled demand for the current period. The Quality of Service Coefficient is the ratio of fulfilled demand to total demand from the last period.
- Step 6: Greedily train the Q-Learning model for the various actions for all the available states by interacting with the above Reinforcement Learning agent with appropriate Quality of Service Index values according to the below formula:

$$Q_{T_{new}}(s,a) = Q_{T_{old}}(s,a) + \alpha * TemporalDifference(s,a)$$

where α is taken to be (1/T), and TemporalDifference(s, a) is defined as:

$$TemporalDifference(s,a) = current \; reward + gamma * \max\left(Q_T(s,a')\right) - Q_{T_{old}}(s,a)$$

where a´ belongs to all available actions for state s, and retrieve the policy learned.

- Step 7: Greedily train the Q-Learning model for the various actions for all the available states by interacting with the above Reinforcement Learning agent without the Quality of Service Index values, i.e., keeping Quality of Service Index =1 and applying the above concepts. Retrieve the policy learned without the Quality of Service Index.
- Step 8: Evaluate the performance of the policies learned in the above two steps along with the performance optimum policy learned using the Value Iteration Method and that of the three standard policies.
- Step 9: We repeat the above steps for different values of Inventory Capacity M and appropriate demand values.

The steps undertaken to arrive at the results for actual demand values are as given below:

- Step 1: Generate the environment values of M, K, alpha, beta, gamma, and delta.
- Step 2: Next, we calculate the rewards for all possible combinations by using:
$$R(s,a,s') = r(s,a,s')\ \{From\ 8\}$$
- Step 3: Design a Reinforcement Learning agent that takes in the current state, action, and Quality of Service Coefficient and returns the current reward, new state, and misses based on the actual demand read from the file. Misses denoting the unfulfilled demand for the current period. The Quality of Service Coefficient is the ratio of fulfilled demand to total demand from the last period.
- Step 4: Greedily train the Q-Learning model for the various actions for all the available states by interacting with the above Reinforcement Learning agent with appropriate Quality of Service Index values according to the below formula:
$$Q_{T_{new}}(s,a) = Q_{T_{old}}(s,a) + \alpha * TemporalDifference(s,a)$$
where α is taken to be (1/T), and TemporalDifference(s, a) is defined as:
$$TemporalDifference(s,a) = current\ reward + gamma * \max\left(Q_T(s,a')\right) - Q_{T_{old}}(s,a)$$
where a´ belongs to all available actions for state s, and retrieve the policy learned.
- Step 5: Greedily train the Q-Learning model for the various actions for all the available states by interacting with the above Reinforcement Learning agent without the Quality of Service Index values, i.e., keeping Quality of Service Index =1 and applying the above concepts. Retrieve the policy learned without the Quality of Service Index.
- Step 6: Evaluate the performance of the policies learned in the above two steps along with the performance optimum policy learned using the Value Iteration Method and that of the three standard policies.
- Step 7: We repeat the above steps for different values of Inventory Capacity M and appropriate actual demand values.

4.2 Data Source

The demand data for testing the algorithm against actual demand was taken from:

https://www.kaggle.com/competitions/walmart-recruiting-store-sales-forecasting/data

4.3 Link for Source Code

The code for the project is available at: https://github.com/PratikMitra5/Multiperiod-Inventory-Management-for-Uncertain-Demand

The repository contains the following two files:

- Multiperiod Inventory Optimization for Actual Demand.ipynb – This file contains code for testing the learning algorithms on random demand with a known probability distribution.
- Multiperiod Inventory Optimization for Random Demand.ipynb – This file contains code for testing the learning algorithms on actual demand values.

5 Results

5.1 Results for random demand

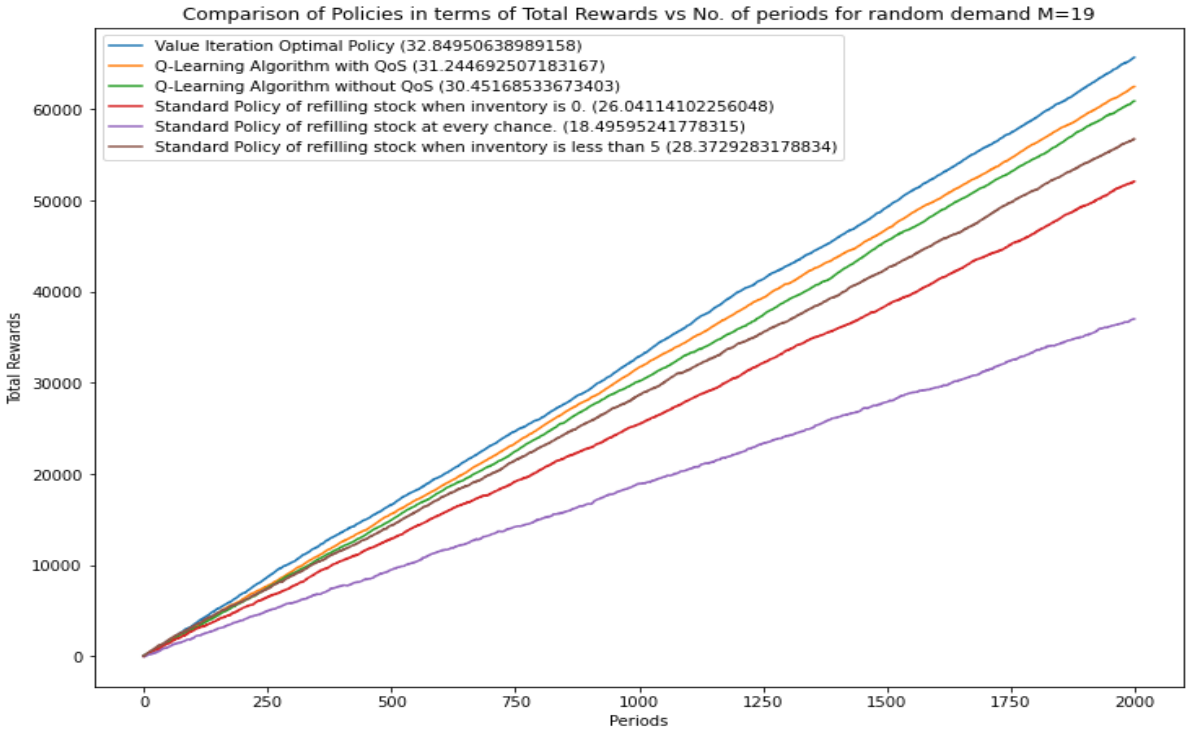

Figure 4 – Graph showing the performance of various policies for random demand and inventory capacity of 19.

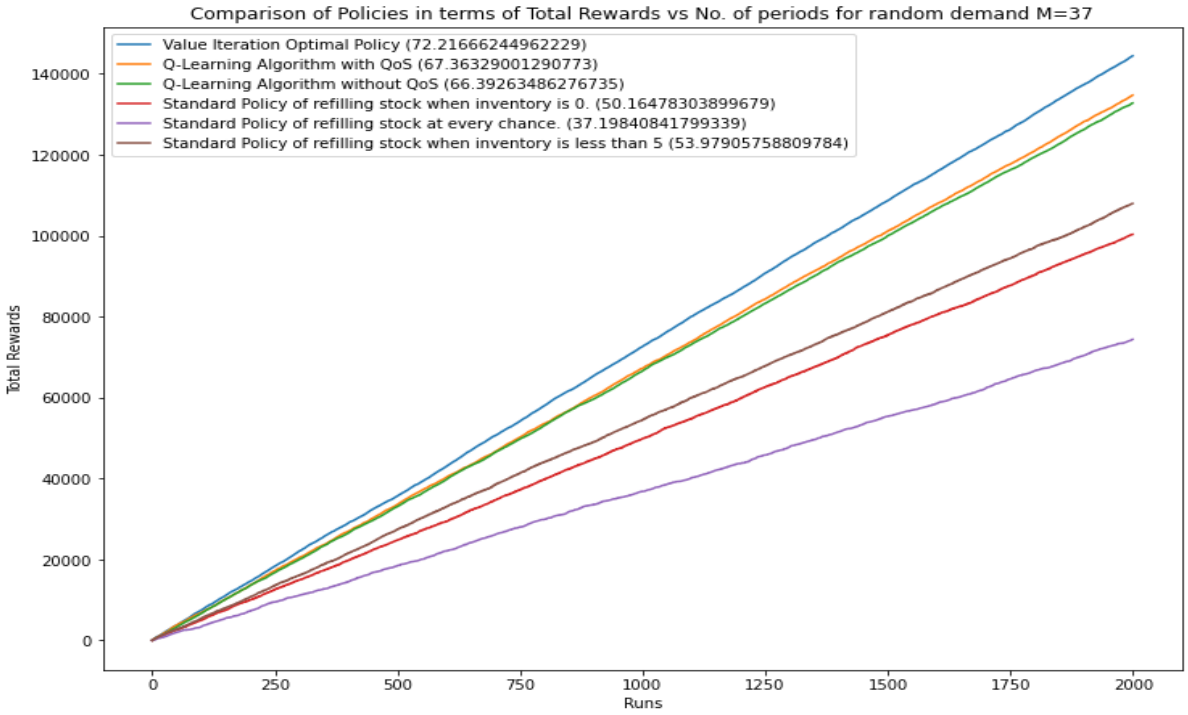

Figure 5 – Graph showing the performance of various policies for random demand and inventory capacity of 37.

We find that when testing the algorithm with random demand, the performance of the learned policies is quite close to the performance of the optimum policy found using the value iteration method. Also, the performance of the learned policies supersedes the performance of standard replenishment policies, and the advantage increases with an increase in the inventory capacity.

5.2 Results for actual demand

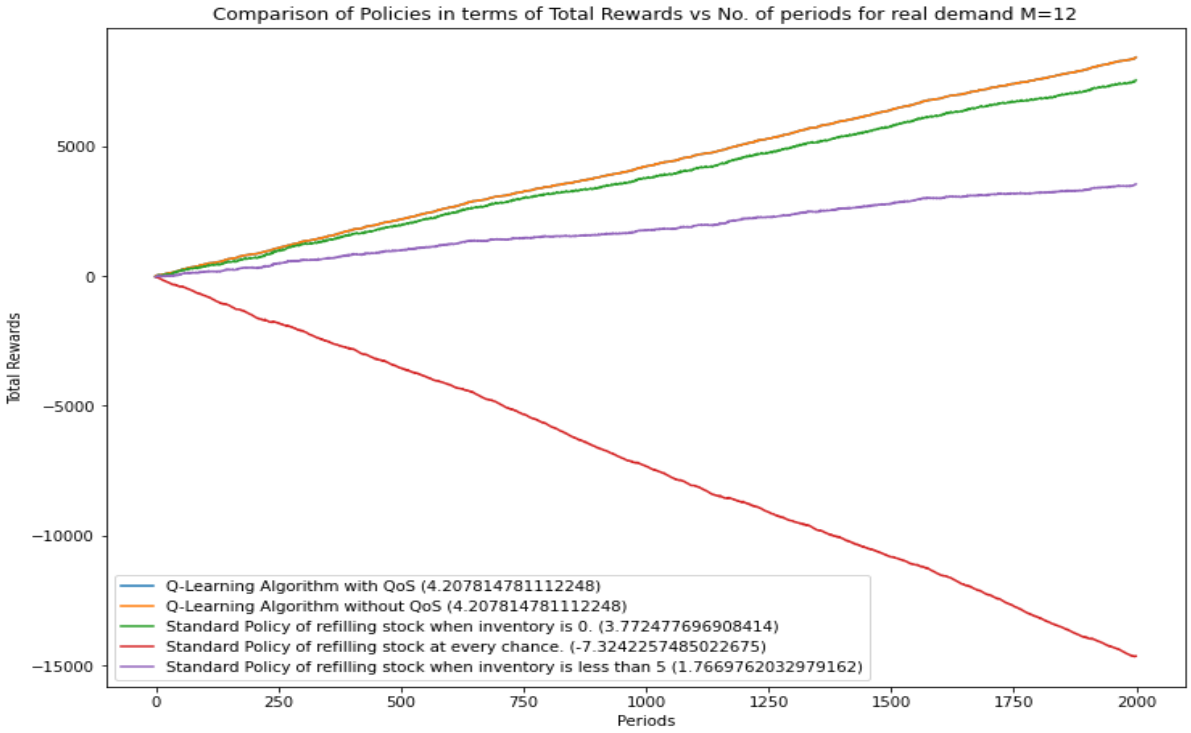

Figure 6 – Graph showing the performance of various policies for actual demand and inventory capacity of 12.

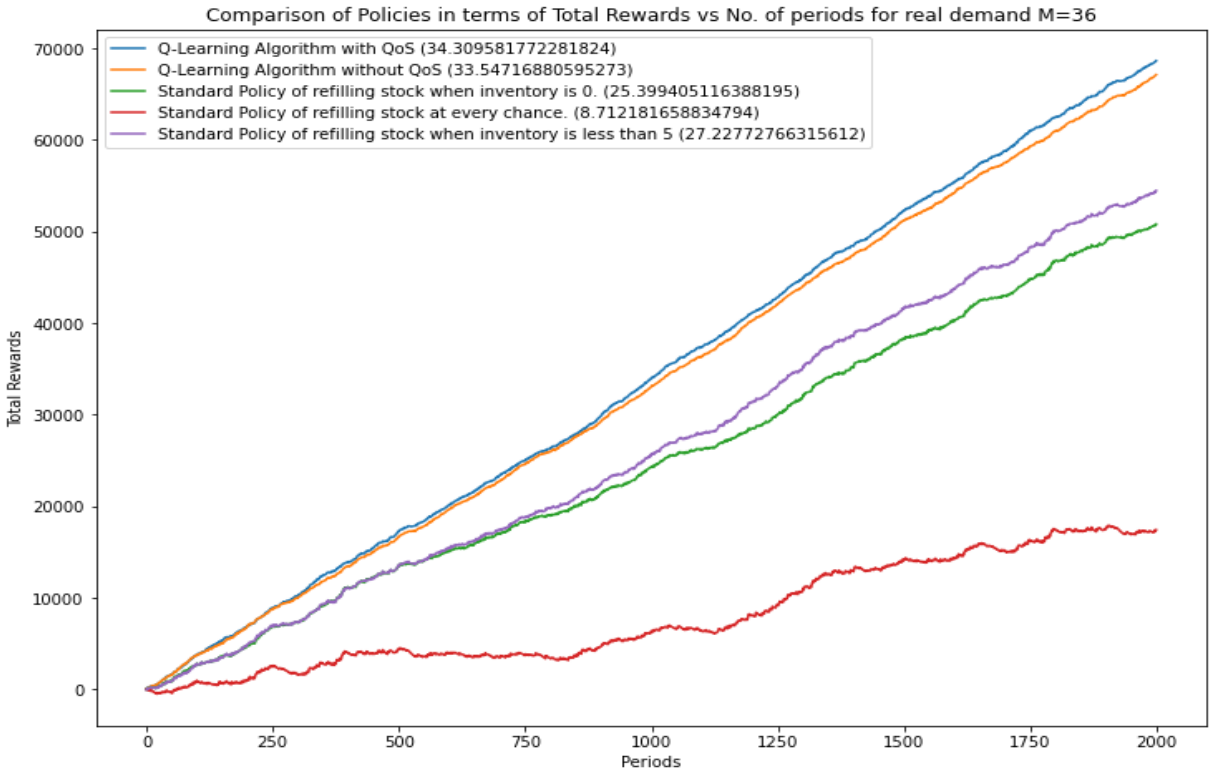

Figure 7 – Graph showing the performance of various policies for actual demand and inventory capacity of 36.

The results show that the Reinforcement learned policies offer a better result than standard policies implemented for taking inventory decisions. Though we see that the improvement for small inventory capacity is comparatively small, we start seeing the advantage of having a Q-Learning algorithm for taking inventory decisions when the inventory capacity increases. We also find that the policy that implements the QoS (Quality of Service) score shows a better result as compared to the one without the QoS implemented.

6 Conclusions and Future Scope

The algorithm performs as expected when tested against random and real demand data. We have also tested the algorithm's robustness by implementing the same for various values of inventory capacity.

We can implement the same concept for managing multi-product inventory models in the future. Another avenue for future improvement is to generalize each of the simplifying assumptions we considered to build the model to build a more generalized model to represent the real-world scenarios better. We can also try to further introduce stochastic lead times and other uncertainties into the model. Also, we can try to model the parameters for specific products like dairy, chocolates, etc. Apart from these, we can test the sensitivity analysis regarding several linear verse Cobb-Douglas demand functions.

7 References

1) Meisheri, Hardik, et al. "A Learning Based Framework for Handling Uncertain Lead Times in Multi-Product Inventory Management." *arXiv preprint arXiv:2203.00885* (2022).

2) Oroojlooyjadid, Afshin, et al. "A deep q-network for the beer game: Deep reinforcement learning for inventory optimization." *Manufacturing & Service Operations Management* 24.1 (2022): 285-304.

3) Gijsbrechts, Joren, et al. "Can deep reinforcement learning improve inventory management? performance on dual sourcing, lost sales and multi-echelon problems." *Manufacturing & Service Operations Management* (2021).

4) Perez, Hector D., et al. "Algorithmic approaches to inventory management optimization." *Processes* 9.1 (2021): 102.

5) Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

6) Goh, M.; De Souza, R.; Zhang, A. N.; He, W.; and Tan, P. 2009. Supply chain visibility: A decision making perspective. In 2009 4th IEEE Conference on Industrial Electronics and Applications, 2546–2551. IEEE.

7) Haijema, R. 2013. A new class of stock-level dependent ordering policies for perishables with a short maximum shelf life. International Journal of Production Economics, 143(2): 434–439.

8) Harifi, S.; Khalilian, M.; Mohammadzadeh, J.; and Ebrahimnejad, S. 2020. Optimization in solving inventory control problem using nature inspired Emperor Penguins Colony algorithm. Journal of Intelligent Manufacturing, 1–15.

9) Yan, Y.; Chow, A. H.; Ho, C. P.; Kuo, Y.-H.; Wu, Q.; and Ying, C. 2021. Reinforcement Learning for Logistics and Supply Chain Management: Methodologies, State of the Art, and Future Opportunities. State of the Art, and Future Opportunities (October 4, 2021). Yang, L.; Li, H.; Campbell, J. F.; and Sweeney, D. C. 2017. Integrated multi-period dynamic inventory classification and control. International Journal of Production Economics, 189: 86–96.

10) Ban G-Y, Rudin C (2019) The big data newsvendor: Practical insights from machine learning. Oper. Res. 67(1):90–108.

11) Chaharsooghi SK, Heydari J, Zegordi SH (2008) A reinforcement learning model for supply chain ordering management: An application to the beer game. Decision Support Systems 45(4):949–959.

12) Clark AJ, Scarf H (1960) Optimal policies for a multi-echelon inventory problem. Management Sci. 6(4):475–490.

13) Claus C, Boutilier C (1998) The dynamics of reinforcement learning in cooperative multi-agent systems. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems, (American Association for Artificial Intelligence, Madison, WI), 746–752.

14) Oroojlooyjadid A, Hajinezhad D (2019) A review of cooperative multi-agent deep reinforcement learning. Preprint, submitted XX, https://arxiv.org/abs/1908.03963.

15) Oroojlooyjadid A, Snyder L, Tak'aˇc M (2017) Stock-out prediction in multi-echelon networks. Preprint, submitted XX, https://arxiv.org/abs/1709.06922.

16) Oroojlooyjadid A, Snyder LV, Tak'aˇcM (2020) Applying Deep Learning to the Newsvendor Problem (Taylor & Francis), 444–463.

17) Pan SJ, Yang Q (2010) A survey on transfer learning. IEEE Trans. Knowledge Data Engrg. 22(10):1345–1359.

18) Pentaho (2008) Foodmart's database tables. Accessed September 30,2015, http://pentaho.dlpage.phi-integration.com/mondrian/mysql-foodmart-database.

19) Snyder LV (2018) Multi-echelon base-stock optimization with upstream stock-out costs. Technical report, Lehigh University, Bethlehem, PA.

20) Snyder LV, Shen Z-JM (2019) Fundamentals of Supply Chain Theory, 2nd ed. (John Wiley & Sons, Hoboken, NJ).

21) Mousavi, S. M.; Hajipour, V.; Niaki, S. T. A.; and Aalikar, N. 2014. A multi-product multi-period inventory control problem under inflation and discount: a parameter-tuned particle swarm optimization algorithm. The International Journal of Advanced Manufacturing Technology, 70(9-12): 1739–1756.

22) Nath, S.; Baranwal, M.; and Khadilkar, H. 2021. Revisiting State Augmentation methods for Reinforcement Learning with Stochastic Delays. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, 1346–1355.

23) Sindhuchao, S.; Romeijn, H. E.; Akc¸ali, E.; and Boondiskulchok, R. 2005. An integrated inventory-routing system for multi-item joint replenishment with limited vehicle capacity. Journal of Global Optimization, 32(1): 93–118.