Final Project: Inventory Management using Reinforcement Learning

1 Introduction

This project addresses the challenge of perishable inventory management using Reinforcement Learning. In industries like food retail, pharmaceuticals, and floral supply chains, products have limited shelf lives, and improper inventory management leads to financial losses from waste or stockouts. This project trains an RL agent to optimize ordering decisions for a perishable product with a 3-day shelf life, balancing revenue from sales against costs of ordering and waste.

Real-World Applications:

- Grocery Stores: Minimize spoilage of fresh produce
- Pharmaceuticals: Manage expiration-sensitive medications.
- E-commerce: Handle seasonal or short-lived products.

The goal of this project is to make the agent learn to order optimally to meet the demand of the product. Prioritize selling near expiration items by giving discount. By setting up the reward structure we are trying to minimize the waste and ordering cost of unnecessary things if there is already product available. We have used Deep Q-network to train the policy of the agent. A Deep Q-Network is basically an extension of Q-learning, a reinforcement learning algorithm, where a deep neural network is used to approximate the Q-value function. This approach allows Q-learning to scale to high-dimensional state spaces.

The Bellman equation for Q-learning is:

$$Q^*(s,a) = r + \gamma \max_{a'} Q^*(s',a')$$

In Deep Q-Networks (DQN), the loss function is defined as:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(y - Q(s,a;\theta))^2\right]$$

where the target Q-value is:

$$y = r + \gamma \max_{a'} Q(s',a';\theta^-)$$

The gradient update for optimizing the neural network parameters θ is:

$$\nabla_\theta L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(y - Q(s,a;\theta))\nabla_\theta Q(s,a;\theta)\right]$$

where: - θ are the parameters of the Q-network, - θ^- are the parameters of the target network, updated periodically, - D is the experience replay buffer.

2 Environment setup

For now we have formulated a simple state and action space for out environment which we will enhance it in the future. We have considered one product in our inventory which the agent learns to manage it and order accodingly. The demand is generated by the possion distribution. The action and state space are given below:

The state is a 3D vector representing inventory counts by remaining shelf life

$$[I_3, I_2, I_1]$$

where:

- I: Items with 3 days left (newly ordered).
- I: Items with 2 days left.
- I: Items with 1 day left (discounted).

As the product age, it is shifted in the state space and if the shelf life of the product is 1 day then the agent automatically initiates discount sale. The idea here is to utilise all the products so that it doesnot go waste. Our reward formulation is based on this concept.

Action space is formulated in a descrite manner. Order quantity (0 to 10 items per day).

The main focus here is the inventory ages by one day and the products are shifted to its respective I1, I2 etc. Items expire daily on I, which is discarded with a waste penalty. The Demand follows a Poisson distribution. where as sales prioritize older stock. The discounted sale value of the product is 5 dollars. while the original selling price is 10 dollar

Reward Function: The reward depends on the revenue of the sales. The Ordering cost is7 dollars per item. Waste penalty is 3 dollars per expired item. The Net Reward is calcuated as:

Revenue - (Ordering Cost + Waste Penalty)

3 Implementation Details

As discussed above, we have used custom gymnasium environment. which extends the gymnasium class. The reset function initialises the inventory to 0.

The step function has multiple things to handle.

- For every step we shift the inventory and apply waste penalty.
- We order new items to the inventory, based on possion distrubution demand is generated. The ordered products are added to I3.
- The reward Calculation is done by combining sales revenue, costs, and penalties.

The DQN architecture consists of three fully connected layers with 64 neurons each that map states to Q-values for each action. To stabilize training, we have used an experience replay buffer which stores 10,000 transitions, and a target network—updated periodically provides consistent Q-value targets, as we know, this is done to eliminate the moving target problem. The agent employs an epsilon greedy strategy with decaying from 1.0 to 0.05 to balance exploration and exploitation. During training, batches of 32 experiences are sampled to compute MSE loss between current and target Q-values, we have used the Adam optimizer with learning rate 1e-3 and =0.99. The agent interacts with the environment for 300 episodes, with rewards penalizing waste, ordering costs, and incentivizing revenue from prioritized sales .The products are discounted near-expiration items first. This setup enables the agent to learn cost aware ordering policies while adapting to stochastic demand using possion distribution. The training is done on 300 episodes and the target network is updated every 10 episodes

4 Results

Below is the plot which demonstrates the training dynamics with reward vs episode comparison.

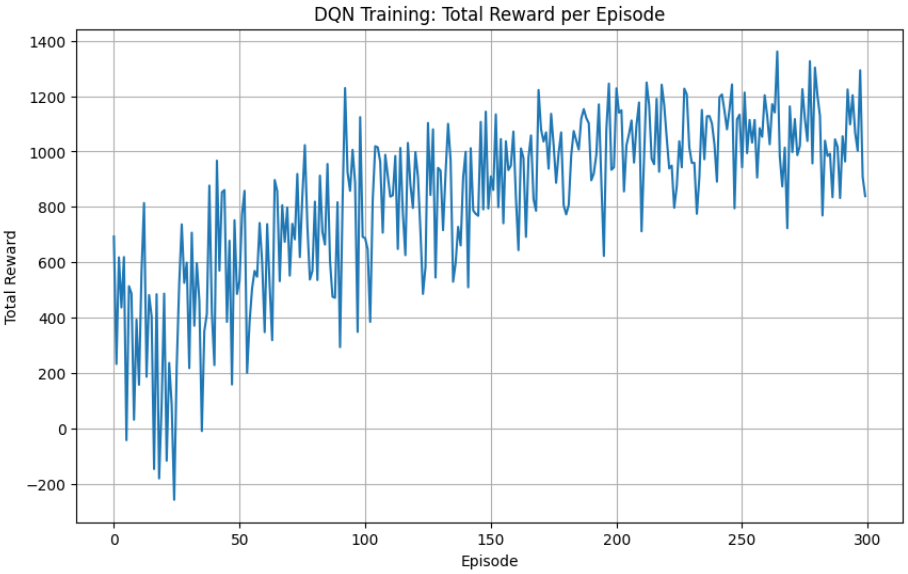


Figure 1: Reward progression in inventory management

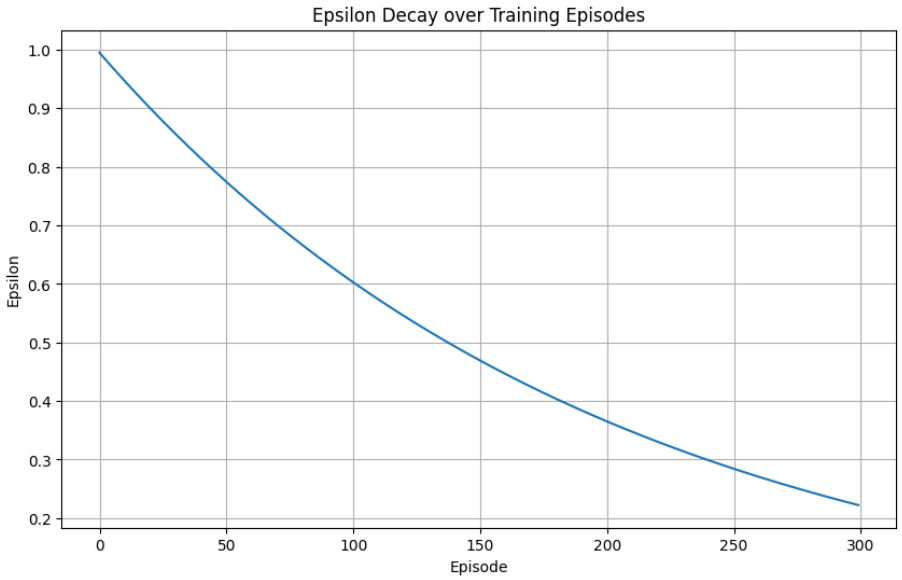


Figure 2: Epsilon change per episode in inventory management

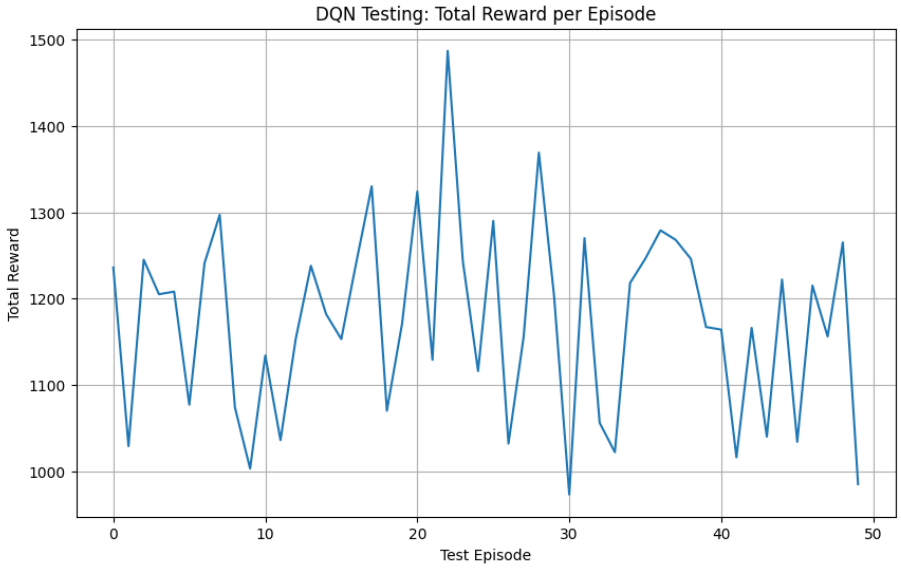


Figure 3: Testing the trained agent. The plot shows rewards per episode

5 Conclusion and Futute work

The agent successfully learned to balance ordering costs, waste penalties, and revenue generation by dynamically adjusting stock levels in response to stochastic demand. Key strategies included prioritizing discounted sales of near-expiration items to minimize waste and maintaining optimal reorder quantities to avoid overstocking. The training and testing results show that the agent is learning to efficiently manage the inventory, but there is room for improvements.

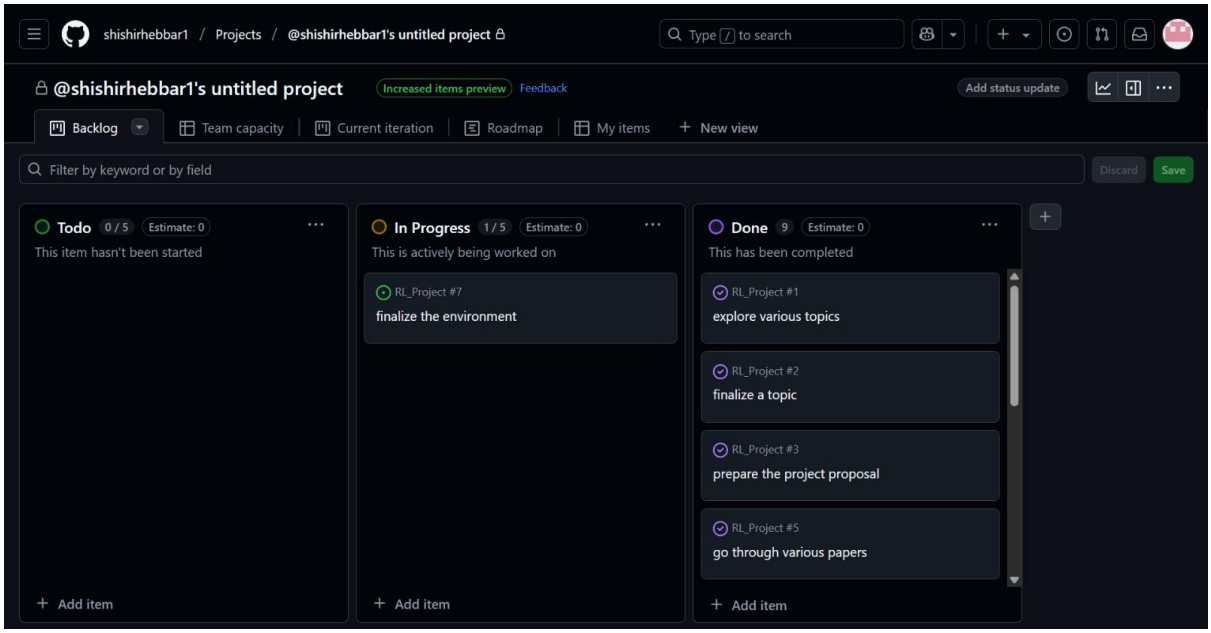
Future work:

- We are planning to use different approach using actor critic and hope to improve the learning even better.
- Currently our environment is designed to handle one product. In future we will make it accomodate multiple products and design the state, action and rewards accordingly.
- Complex demand ordering can be done to take into consideration of promotional sales and seasonal trends etc.

Github Link for our Repo:

- https://github.com/shishirhebbar1/RL_Project

Git Project Board:



References:

- https://gymnasium.farama.org/
- https://matplotlib.org/
- https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf
- Understanding Dueling DQN: A Deep Dive into Reinforcement Learning | by Jagjit Saini | Medium