Project Report

Reinforcement Learning for Inventory Management

Developed by

VASEEGARAN B

1.Introduction:

Inventory management balances holding costs and stockout risks. Traditional rule-based methods may fail under complex demand patterns. Reinforcement Learning (RL) provides a data-driven approach to learn optimal policies by interacting with a simulated environment. This project applies RL using the M5 Forecasting dataset, Gymnasium environments, Stable-Baselines3 agents (PPO and DQN), and Python-based data processing and visualization.

2.RL Formulation:

1. Reward Function

Formula:

$$\text{base\_reward}_t = \alpha_{\text{service}} \cdot (\text{served\_units}_t \cdot \text{sell\_price}_t) - \beta_{\text{hold}} \cdot \text{inventory\_level}_t - \gamma_{\text{stockout}} \cdot \text{unmet\_units}_t$$

$$\text{reward}_t = 5 \cdot \tanh\left(\frac{\text{base\_reward}_t}{\text{scale}} - 0.01 \cdot |\text{order\_qty}_t - \text{last\_order\_qty}_{t-1}|\right)$$

2. Inventory Update

Formula:

$$\text{inventory\_level}_{t+1} = \min(\text{inventory\_level}_t + \text{order\_qty}_t, \text{max\_inventory}) - \text{sales}_t$$

3. Sales (served units)

Formula:

$$\text{sales}_t = \min(\text{inventory\_level}_t + \text{order\_qty}_t, \text{demand}_t)$$

4. Service Level

Formula:

$$\text{Service Level} = \frac{\sum \text{served units}}{\sum \text{demand} + \epsilon}$$

5. Profit

Formula:

$$\text{Profit}_{\text{episode}} = \sum_t \text{reward}_t$$

6. Action Clamping

Formula:

$$\text{sales}_t = \min(\text{inventory\_level}_t + \text{order\_qty}_t, \text{demand}_t)$$

7. Observation State

Formula:

$$\text{obs}_t = [\text{inventory\_level}_t, \text{recent\_demand}_{t-6:t}, \text{day\_of\_week}_t, \text{promo\_flag}_t]$$

The inventory management problem is formulated as a Markov Decision Process

2.1 State Space

The state at time 't' is represented as:

$$s_t = [\text{inventory level}_t, \text{demand}_{t-6..t}, \text{day\_of\_week}_t, \text{promo\_flag}_t]$$

- Demand history (7-day window): Provides context for the agent.
- Day of week: Encodes seasonality effects.
- Promo flag: Indicates if a promotion affects demand.
- Inventory level: Current units in stock.

2.2 Action Space

The agent can select a discrete order quantity:

$$a_t \in \{0, 1, 2, \ldots, \text{max\_order}\}$$

max_order is dynamically set based on historical demand quantiles or manually specified.

2.3 Reward Function

The reward captures multiple objectives: meeting demand, minimizing holding costs, and penalizing stockouts. The hybrid reward formula:

$$\text{base\_reward}_t = \alpha_{\text{service}} \cdot (\text{served\_units}_t \cdot \text{sell\_price}_t) - \beta_{\text{hold}} \cdot \text{inventory\_level}_t - \gamma_{\text{stockout}} \cdot \text{unmet\_units}_t$$

$$\text{reward}_t = 5 \cdot \tanh\left(\frac{\text{base\_reward}_t}{\text{scale}} - 0.01 \cdot |\text{order\_qty}_t - \text{last\_order\_qty}_{t-1}|\right)$$

- α,β,γ are weights for service, holding, and stockout penalties.
- Scale normalizes rewards to prevent instability.
- Smoothness penalty discourages drastic order changes.
- Reward is squashed using tanh for stability.

3. RL Algorithms Used

- Proximal Policy Optimization (PPO)
- Deep Q-Network (DQN)

3.1 Proximal Policy Optimization (PPO)

- Type: On-policy, policy-gradient algorithm.
- Advantage: Stable updates with clipping, suitable for continuous and discrete action spaces.
- **Objective:**

$$\max_\theta \; L^{CLIP}(\theta)$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big]$$

3.2 Deep Q-Network (DQN)

- Type: Off-policy, value-based algorithm.
- Advantage: Handles discrete action spaces efficiently.
- Objective: Minimize temporal difference (TD) error:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

Implementation: Using Stable-Baselines3 for both PPO and DQN.

4. Environment Setup

4.1 Data Processing

- Dataset: M5 Forecasting Accuracy (sales, calendar, sell prices).
- Preprocessing steps:
  1).Filter by item_id and store_id to reduce memory load
  2).Melt sales data to long format (one row per day).
  3).Merge calendar and sell prices.
  4).Fill missing values and encode categorical features.
  5).Compute rolling features:
    - 7-day rolling average of demand.
    - 30-day rolling average of sell price.

4.2 Gymnasium Environment

- InventoryEnv inherits from gym.Env.
- Methods:
  - reset(): Initializes inventory and step counter.
  - step(action): Updates inventory, computes reward, and returns next state.
  - render(): Prints current inventory and total reward.

Key dynamics:

- Orders are instantly delivered.
- Sales limited by available inventory.
- Reward combines revenue, holding costs, stockout penalty, and action smoothness.

5. Experiments

5.1 Training Setup

- Agents: PPO and DQN.
- Environment: Filtered M5 item/store data.
- Timesteps: 50,000 (or configurable).
- Vectorized environment: DummyVecEnv for Stable-Baselines3 compatibility.

5.2 Evaluation Metrics

- Average Reward: Cumulative reward per episode.
- Service Level: Fraction of demand served.

$$\text{service level} = \frac{\sum_t \text{served units}}{\sum_t \text{demand}}$$

- Profit: Equal to cumulative reward in this formulation.
- Visualization: Inventory, demand, and reward over time with reorder triggers.

5.3 Visualization

- Inventory plotted as a step line.
- Reorder markers indicate threshold triggers.
- Demand and reward plotted with raw and smoothed lines (rolling mean).

6. Challenges and Solutions

Challenge	Description	Solution
High memory usage	Full M5 dataset is too large for training	Filter by specific `item_id` and `store_id`
Sparse rewards	Many steps have zero sales, causing slow learning	Hybrid reward function combines revenue, stockout, and smoothness penalties
Inventory constraints	Orders may exceed max inventory	Clamping action: `inventory_level = min(inventory + order, max_inventory)`
Non-stationary demand	Daily demand varies with trends and promotions	Include 7-day demand window, day-of-week, and promo flags in state
RL stability	DQN and PPO can be unstable	Normalization of rewards using scaling and `tanh` squashing; vectorized environment for PPO

7. Conclusions

- RL can effectively learn inventory policies that balance service levels and holding costs.
- PPO and DQN both perform well, but PPO tends to be more stable due to policy gradient clipping.
- Using rolling features, day-of-week, and promotion flags improves the agent's ability to anticipate demand fluctuations.
- Visualization provides actionable insights on inventory dynamics and reorder decisions

Github Repo link : https://github.com/vasee-garan/Inventory_Management_system.git