Using General Value Functions to Learn Domain-Backed Inventory Management Policies

Durgesh Kalwar
TCS Research
Mumbai, India
durgesh.kalwar@tcs.com

Omkar Shelke
TCS Research
Mumbai, India
shelke.omkar@tcs.com

Harshad Khadilkar
TCS Research
Mumbai, India
harshad.khadilkar@tcs.com

ABSTRACT

We consider the inventory management problem, where the goal is to balance conflicting objectives such as availability and wastage of a large range of products in a store. We propose a reinforcement learning (RL) approach that utilises General Value Functions (GVFs) to derive domain-backed inventory replenishment policies. The inventory replenishment decisions are modelled as a sequential decision making problem, which is challenging due to uncertain demand and the existence of aggregate (cross-product) constraints. In existing literature, GVFs have primarily been used for auxiliary task learning. We use this capability to train GVFs on domain-critical characteristics such as prediction of stock-out probability and wastage quantity. Using this domain expertise for more effective exploration, we train an RL agent to compute the inventory replenishment quantities for a large range of products (up to 6000 in the reported experiments), which share aggregate constraints such as the total weight/volume per delivery. Additionally, we show that the GVF predictions can be used to provide additional domain-backed insights into the decisions proposed by the RL agent. Finally, since the environment dynamics are fully transferred, the trained GVFs can be used for faster adaptation to vastly different business objectives (for example, due to the start of a promotional period or due to deployment in a new customer environment).

KEYWORDS

Inventory management, General value functions, Reinforcement learning, Scalability

1 INTRODUCTION

The Inventory Management (IM) problem is one of the most critical challenges within the supply chain industry [24], particularly when it comes to replenishment decisions. Striking the right balance between maintaining adequate stock levels to meet the customer demands while minimizing inventory costs is a perpetual concern. Achieving this delicate equilibrium has profound implications for cost efficiency, customer satisfaction and overall supply chain performance. In the context of supply chain, the management of replenishment decisions is pivotal since it directly influences inventory turnover, lead times, and the ability to respond swiftly to fluctuations within demand. Efficient and data-driven strategies address this complex issue effectively and they continue to be a subject of rigorous research and practical innovation within the industry. Some of the formative work in this domain employing dynamic programming [8], provided a theoretical foundation for optimal base-stock policies in simple series systems. Nonetheless, the complexity of the recursive equations grows substantially with the problem's size, prompting the development of various approximation algorithms, such as stochastic approximation [21], infinitesimal perturbation analysis (IPA) [27], and approximate dynamic programming [4], to tackle this. However, the practical implementation of these methods remains a formidable challenge [5].

Traditional methods in inventory management (IM) have often simplified the problem by applying dynamic programming (DP) techniques. But, such approaches tend to make assumptions that may not hold in real-world scenarios. For instance, they often assume that customer demands are independently and identically distributed (iid) and that leading times are deterministic [11, 19]. These assumptions, while convenient for mathematical modeling, can deviate significantly from actual conditions. Furthermore, as certain critical factors like leading times and the number of stock keeping units (SKUs) increase, the state space of the problem expands rapidly. This expansion, often referred to as the "curse of dimensionality" [13], renders the problem intractable when addressed using DP. To overcome these limitations, various approaches based on approximate dynamic programming have been introduced to address IM challenges across diverse contexts [7, 12, 14]. While these alternative methods have demonstrated efficacy in specific scenarios, they tend to rely heavily on problem-specific expertise or make assumptions that might not be universally applicable. For instance, some of these approaches assume zero or one-period leading times [14], which might not hold true in every inventory management setting. Consequently, their ability to generalize to broader IM contexts remains constrained.

Reinforcement Learning (RL) algorithms have proven effective in a wide array of applications involving sequential decision-making. The applications encompass areas such as software-based gaming [10, 29], as well as physical systems like autonomous driving [28], and flight dynamics [26]. A notable characteristic of these applications is their capacity to devise strategies that optimize long-term future rewards while adhering to constraints. This quality positions RL as a natural choice for a broader category of problems. We investigate its applicability in high-dimensional systems, particularly in real-world contexts like industrial operations. Our focus is on systems driven by vector differential equations, stemming from operations research. However, this approach can be adapted to any scenario requiring complex decision-making without online searching. RL methods are versatile for data-driven situations. Yet, creating a single policy for all stock keeping units (SKUs) is challenging due to the vast state and action space involved [16]. Two recent papers [2, 30] where more realistic scenarios containing multiple SKUs are considered. In [2], the main contribution is to propose a framework to support efficient deployment of RL algorithms in real systems. As an example, the authors introduce

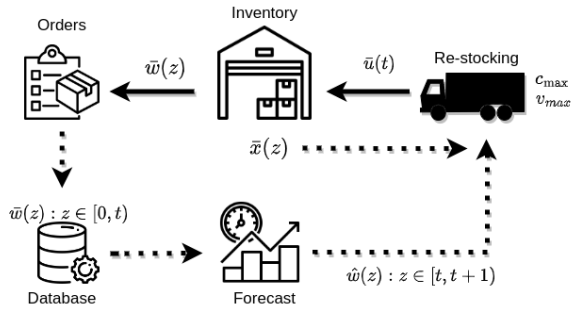

Figure 1: A simple flowchart of the inventory management problem at hand

a centralized algorithm for solving the IM problem. In contrast, a decentralized algorithm is proposed in [30] to solve the IM problem with multiple SKUs in a multi-echelon setting. These two papers use the off-the-shelf RL algorithm [9] to solve for the policy.

General Value Functions (GVFs) are crucial in reinforcement learning (RL), offering a versatile framework [31] for estimating various value functions, from standard state-value and action-value functions to specialized RL objectives. GVFs empower agents to grasp the value of different signals or goals in an environment, enhancing their understanding of diverse aspects of the learning problem. They prove especially valuable for abstracting and generalizing information across tasks and scenarios, enabling efficient knowledge transfer, which is vital in complex, dynamic environments where a single value function may fall short. Recent RL advancements, like the Option-Critic Architecture [1] and successor representation [20], rely on GVFs to boost learning efficiency across diverse tasks. In a recent study [18], GVFs were applied in navigation environments with vast state spaces and sparse rewards. They were employed for learning auxiliary tasks as value functions and generating directed exploration actions.

In this paper, we aim to showcase the following contributions:

(1) We present an RL strategy which makes use of General Value Functions for learning the underlying dynamics of an inventory management problem, thus improving (i) the effectiveness of exploration during training, and (ii) the transferability of learnt embeddings and policies to other tasks within the same environment. Our approach is specifically trained on system attributes such as critical levels of inventory and wastage quantity generated.
(2) Experiments on scenarios ranging from 100 to 6000 products are carried out along with other baselines. We provide further insights into the GVF learnt models by portraying their predictions for different attributes. Also we showcase their generalization abilities by their quick assimilation to other domain objectives.

2 PROBLEM DESCRIPTION

We consider a retail store that offers a variety of P products (Figure 2), and the inventory of each product is denoted as $x_i(t)$ at time step t (where $1 \leq i \leq P$), is closely monitored. The store has the flexibility to request an order in each time step, specifying a quantity, $u_i(t)$ to be replenished from the warehouse. Each product is subject to a maximum storage shelf limit, M_i. Both x_i and u_i are scaled variables, calculated by dividing the actual product quantity by M_i. We assume that the time gap between placing an order and receiving the replenishment is negligible compared to the duration of each time step. Given these consideration, the state is updated as follows:

$$\mathbf{x}(t)^+ = \mathbf{x}(t)^- + \mathbf{u}(t) \tag{1}$$

We have P-sized vectors denoted as $\mathbf{x}(t)^-$, $\mathbf{x}(t)^+$, and $\mathbf{u}(t)$, which pertain to all the products. These vectors are scaled based on the maximum inventory capacity for each respective product. The superscripts + and − signify the inventory level immediately prior to and following replenishment. The store's limitations are defined by the following constraints:

$$\mathbf{0} \leq \mathbf{x}(t) \leq \mathbf{1}, \tag{2}$$

$$\mathbf{0} \leq \mathbf{u}(t) \leq \mathbf{1} - \mathbf{x}(t)^-, \tag{3}$$

$$\mathbf{v}^\intercal \, \mathbf{u}(t) \leq v_{\max} \quad \text{and} \quad \mathbf{c}^\intercal \, \mathbf{u}(t) \leq c_{\max}. \tag{4}$$

In this context, constraint (2) places a restriction on the normalized inventory level for each individual product. Constraint (3) specifies that the amount of products requested during any given moment must not surpass the remaining shelf capacity for those specific products at that time. Constraint (4) establishes an upper limit, denoted as $v_{\max}$, on the aggregated volume, and $c_{\max}$ on the aggregated weight, considering all the products, respectively. This constraint effectively accounts for the limitations to transportation capacity. The relationship between inventory levels from on time step to the next is described as follows:

$$\mathbf{x}(t+1)^- = \max\left(\mathbf{0}, \mathbf{x}(t)^+ - \mathbf{w}(t, t+1)\right), \tag{5}$$

Here, $\mathbf{w}(t, t+1)$ represents the vector indicating the quantities requested for each product between time steps t and $(t+1)$. The objective of the replenishment algorithm is to calculate the ideal replenishment quantity, denoted as $\mathbf{u}(t)$ which aims to maximize a 'business reward' based on the following explanation.

Inventory management poses a multi-objective optimization challenge, involving explicit costs associated with two primary aspects: (1) the risk of depleting inventory to zero, commonly referred to as "out-of-stock," and (2) the quantity $q_{waste,i}(t)$ of products that become wasted or spoiled during the time period ending at t. Additionally, we aim to ensure equitable treatment among products, even when the system faces stress, such as when capacities like $v_{\max}$ and $c_{\max}$ cannot adequately keep pace with product demand. To address this, we introduce a fairness penalty that considers the variation in inventory levels across all products, spanning from the 95th to the 5th percentile, denoted as $\Delta\mathbf{x}(t)_{.05}^{.95}$. Furthermore, we incorporate a penalty term $\Omega(t)$, which encompasses the total refused orders during the specified time interval, aggregating across all products. The objective, which we seek to maximize, is defined in equation (6). Given that no control interventions are allowed between consecutive time steps, all terms can be considered as aggregate values received at time t. The objective function is

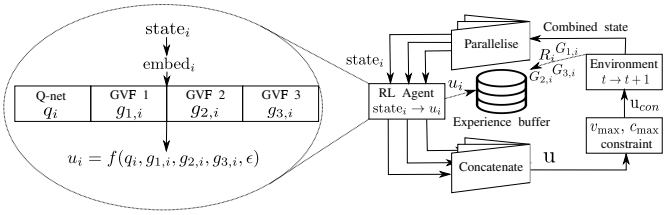

Figure 2: Implementation of DEZ-Greedy in the proposed problem. Copies of the RL agent process the individual state$_i$ **for each product** i**.**

expressed as follows:

$$R(t) = 1 - \underbrace{\frac{p_{\text{empty}}(t)}{p}}_{\text{Out of stock}} - \underbrace{\frac{p_{\text{critical}}(t)}{p}}_{\text{Critical level}} - \underbrace{\frac{\sum_i q_{\text{waste,i}}(t)}{p}}_{\text{Wastage}} - \underbrace{\Delta \mathbf{x}(t)_{.05}^{.95}}_{\text{Percentile spread}} - \underbrace{\Omega(t)}_{\text{Refused orders}} \tag{6}$$

where p represents the total count of products (corresponding to the size of $\mathbf{x}$, $p_{\text{empty}}(t)$ signifies the quantity of products with $x_i = 0$ at the conclusion of time interval $[t-1, t)$. $p_{\text{critical}}(t)$ is the number of products with $x_i < \kappa_i$ at the end of the same time interval, ensuring that each product maintains at least its minimum presentation level κ_i or a critical inventory level. Since the highest possible value for $p_{\text{empty}}(t)$, $p_{\text{critical}}(t)$ and $\Delta(t)$ equals p, the maximum potential value for $q_{\text{waste,i}}(t-1, t)$ is 1. Consequently, the theoretical range for the objective/reward lies within the interval of -4 to 1. For practical purposes, individual terms are expected to be smaller than 1, and the majority of the reward should fall in the range of -1 to 1. The primary aim of the algorithm is to optimize the discounted cumulative sum of this reward defined in equation (6), while adhering to the constraints detailed in equations (2) to (4).

3 METHODOLOGY

Our solution makes use of DQN [23] which is an popular off-policy algorithm where we use the previous policy samples to improve it's value estimation. Along with DQN we define GVFs to estimate a domain-critical property and use them as sub-policies to learn predictive knowledge of the system. We also utilize the Directed EZ-greedy strategy [18], as another baseline. We compare these different algorithms for our study and show their performance across different number of products.

We note that the order rate $\mathbf{w}(\mathbf{z})$ plays a key role in the system dynamics (4). For simplicity, we define an estimator for the sales rate w_i of each product i in the form of a trailing average of sales in the most recent T time periods,

$$\hat{w}_i(z) = \frac{\int_{t-T}^{t} w_i(z')\, dz'}{T}, \forall z \in [t, t+1) i \in 1, ..., p \tag{7}$$

It must be remarked that the above forecasting algorithm returns a very crude approximation of the actual demand. Instead, one can leverage more sophisticated forecasting algorithms [3, 6, 15], such as recurrent neural network(RNN)-based approaches to provide a better estimate of the true demand; however, design of the most appropriate forecasting algorithm is not the main focus of this paper. Instead, we rely on the simpler averaging based forecasting algorithm and still achieve near-optimal performance for inventory control. A better forecasting algorithm would only aid the RL algorithm in further improving its capability towards optimal replenishment decision. Since we assume that each period lasts for a unit of time, the forecast for aggregate orders, $W_i(t)$, is also given by (7).

The primary challenges in applying RL to this problem are (a) the large number of products p, (b) handling the shared capacity constraints (4), and (c) the fact that the number of products can change over time. We describe an algorithm for parallelised computation of replenishment decisions, by cloning the parameters of the same RL agent for each product and computing each element of the vector $\mathbf{u}(\mathbf{t})$ independently. The advantage of this approach is that it splits the original problem into constant-scale sub-problems. Therefore, the same algorithm can be applied to instances where there are a very large (or variable) number of products. Despite parallelisation, we handle the shared capacity constraints as follows.

3.1 Rewards

The computation of individual components of $\mathbf{u}$ faces two primary difficulties: firstly, guaranteeing compliance with the system-level constraints outlined in equation (4), and secondly, ensuring equitable treatment of all products. These challenges are mitigated to some extent through the reward system. The fairness concern is specifically addressed by incorporating the percentile spread term mentioned in equation (6), as it penalizes the agent when certain products have low inventory levels while others are maintained at higher levels. Moreover, the constraints related to volume and weight are introduced as flexible penalties in the subsequent definition of the 'per-product' reward, which is customized for individual decision-making purposes.

$$\begin{aligned} R_i(t) = 1 - b_{i,empty}(t) - b_{i,critical}(t) - q_{waste,i}(t) \\ -\Delta \mathbf{x}(t)_{.05}^{.95} - \Omega_i(t) - \alpha \max(\rho - 1, 0) \end{aligned} \tag{8}$$

where $b_{i,empty}(t)$ and $b_{i,critical}(t)$ are binary variables that serve as indicators, determining whether the inventory for product i falls below 0 or κ_i in the current time period. Additionally, α stands as a fixed parameter, while ρ represents the ratio of the total volume or weight requested by the RL agent to the existing capacity. This can be precisely defined as follows:

$$\rho = \max\left(\frac{\mathbf{v}^\top \mathbf{u}(\mathbf{t})}{v_{max}}, \frac{\mathbf{c}^\top \mathbf{u}(\mathbf{t})}{c_{max}}\right)$$

Equation (8) outlines the reward provided to the RL agent, which differs from the genuine system reward described in (6). When the combined actions generated by the agent for all products do not surpass the available capacity ($\rho \leq 1$), then the average value of (8) equals that of (6). This suggests that maximizing $R_i(t)$ is essentially the same as maximizing $R(t)$, as long as the system constraints remain unbroken. The final two components of (8) are consistent for all products at the given time t.

3.2 State and action space

Table 1 presents a list of features utilized to calculate the replenishment quantity for each product. The initial two features pertain

Notation	Description
$x_i(t)$	Current inventory level
$\hat{W}_i(t)$	Forecast aggregate orders in $[t, t+1)$
v_i	Unit volume
c_i	Unit weight
$T_s(i)$	Shelf-life
$\mathbf{v}^\intercal \hat{\mathbf{W}}(\mathbf{t})$	Total volume of forecast for all products
$\mathbf{c}^\intercal \hat{\mathbf{W}}(\mathbf{t})$	Total weight of forecast for all products

Table 1: State space representation

to the current state of the system concerning product i. The subsequent three inputs contain product-related information, encompassing both long-term and unchanging behavior. The parameter shelf-life $T_s(i)$, is a normalized measure inversely proportional to the average inventory reduction for product i signifying the decrease in inventory not explained by orders during a specified time frame. These metadata elements are used to distinguish between different product characteristics when they are processed sequentially by the same RL agent, achieved by mapping individual products into a common feature space. The final two features listed in Table 1 are derived indicators that offer insights into the overall demand on the system, considering various constraints. These indicators function as constraints on the control action for product i when the system experiences high demand. They also aid the agent in establishing a connection between the last term in the observed rewards (the penalty for exceeding capacity) and the input states. The outcome of the RL agent is $u_i(t)$ which represents the intended action for product i at time t Individual actions are combined to form $\mathbf{u}(\mathbf{t})$ as depicted in the workflow illustrated in Figure 1.

3.3 General Value Functions

The general value function is a more generalized definition of the value function in reinforcement learning. It can be learned for any predictive signal not just for reward signal. In literature, these predictive signals are also known as cumulants C. These cumulants can be thought of as queries directed towards GVFs. Formally, GVFs are defined as the expected cumulative discounted sum of cumulants for a given policy π and a state-action pair (s, a):

$$Q^{GVF}(s_t, a_t) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t C_t(s_t, a_t)] \tag{9}$$

Where γ is a discount factor. And GVFs can be learned by any value-based RL algorithm e.g. temporal difference (TD) learning. For this environment setup we used 3 GVFs which are trained on following cumulant signals defined as -

$$C_{1,i}(t) = q_{waste,i}(t) \tag{10}$$

where, $q_{waste,i}(t)$ is the wastage quantity of product i during the time period end at t. The GVF learned on cumulant signal $C_{1,i}(t)$, we denote it by Q^{GVF_1} and its policy by $\pi^{GVF_1} = \underset{a}{\text{argmin}}\, Q^{GVF_1}(s, a)$.

$$C_{2,i}(t) = \begin{cases} 1, & \text{if } x_i(t) = 0. \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Cumulant $C_{2,i}(t)$ represents the product i going out of stock at the end of time-period $[t-1, t)$. The GVF learned on cumulant signal $C_{2,i}(t)$, we denote it by Q^{GVF_2} and its policy by $\pi^{GVF_2} = \underset{a}{\text{argmin}}\, Q^{GVF_2}(s, a)$.

$$C_{3,i}(t) = 1 - x_i(t) \tag{12}$$

Cumulant $C_{3,i}(t)$ encodes the depletion in inventory of product i during the time period end at t. The GVF learned on cumulant signal $C_{3,i}(t)$, we denote it by Q^{GVF_3} and its policy by $\pi^{GVF_3} = \underset{a}{\text{argmin}}\, Q^{GVF_3}(s, a)$.

3.4 Directed EZ-Greedy Strategy

Directed EZ Greedy utilizes General Value Functions (GVFs) as auxiliary tasks and employs sub-policies derived through greedy action selection for a temporally extended ϵ-greedy approach. It occasionally chooses a random option (with probability ϵ), possibly originating from any of the randomly selected GVFs in the current state. It then continues to take the greedy action based on that GVF, facilitating directed exploration across various state regions. This distinctive exploration strategy is why it's called Directed EZ Greedy. Algorithm 1 is adapted from [18] and in this paper we kept the maximum persistence to 1.

Function `DEZGreedy(`ϵ`)`**:**
- Selected GVF index $g \leftarrow 0$
- **while** *True* **do**
  - Observe state s
  - **if** $random() < \epsilon$ **then**
    - // Explore
    - Sample GVF: $g \sim [0, M]$
    - **if** $g == 0$ **then**
      - Sample action: $a \leftarrow U(A)$
    - **else**
      - $a \leftarrow argmin(Q_g^{GVF})$
    - **end**
  - **else**
    - // Exploit
    - $a \leftarrow argmax(Q^{Main})$
  - **end**
  - Take action a
- **end**

Algorithm 1: DEZ-Greedy exploration strategy

4 EXPERIMENTAL DETAILS

4.1 Data for experiments

As our methodology aligns with [22], we use a similar procedure to generate out data. [22] conducted experiments on publicly available data set for brick and mortar stores [17]. The original data set encompasses purchase information for 50,000 different product types and involves 60,000 unique customers. However, it lacks metadata about products such as dimensions and weight. To align this data with the format required for their work, they attributed dimensions and weights to each product type based on the product descriptions available. This process was carried out manually, so [22] only used two distinct subsets containing 220 and 100 products, respectively,

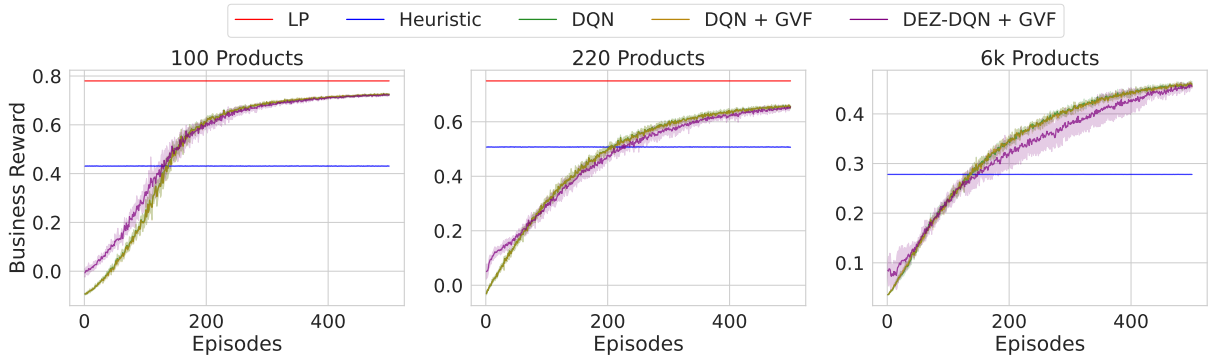

Figure 3: Comparing algorithm training performance on varying dataset sizes: Mean results across 5 random seeds with 95% confidence interval shading for 100, 220, and 600 product datasets

from the complete data set. We formulated an algorithmic approach to generating and assigning metadata, allowing us to extend the experiments up to 6000 products and we did the experiments on all three datasets containing 6k, 220 and 100 products. These subsets are specifically chosen from products categorized as 'grocery' from original data set [17].

All three datasets covers a span of 349 days and we assume that stock replenishment occurs four times daily, resulting in a total of 1396 time periods. From these, we allocate the initial 900 time periods for training purposes and reserve the remaining 496 time periods for testing. We utilize three semi-synthetic data sets, each consisting of 6k, 220 and 100 product types, to compare against other approaches. We set the volume and weight constraints (denoted as vmax and cmax) slightly below the average sales volume and weight to ensure that constraints 4 remain active.

For RL based algorithms, we discretized the replenishment values for each product i in 14 discrete actions (0, 0.005, 0.01, 0.0125, 0.015, 0.0175, 0.02, 0.03, 0.04, 0.08, 0.12, 0.2, 0.5, 1). These actions represent the normalized replenishment quantities relative to the maximum shelf capacity for each respective product. During each training episode, we generate random initial inventories for each product. These random initial inventories are used for all the experiments across algorithms (RL, heuristic, linear programming). We ran experiments for 5 random seeds for statistical significance.

4.2 Baseline algorithms

4.2.1 ***Heuristic based on proportional control:*** We employ a customized version of the s-policy [25], a well-established heuristic in the literature aimed at maintaining inventories at predefined constant levels. In this heuristic, denoting x^* as the target inventory level for products, the replenishment quantity is carefully designed to meet forecasted sales and bring the inventory to the desired level x^* by the end of the time period. This is mathematically expressed as:

$$u_{pr}(t) = \max[0, x^* + \hat{W} - x(t)^-] \tag{13}$$

The action derived from Eq. (13), denoted as $u_{pr}(t)$, satisfies constraints (2) and (3) since $0 \leq x^* \leq 1$. However, it is important to note that this approach does not guarantee compliance with the total volume and weight constraints specified in Eq. (4).

4.2.2 ***Optimal bounds using linear programming:*** In this paper, we employ the same formulation as in [22] to construct an online linear programming (LP) model with perfect information, where the true or realized demands W are known beforehand. The LP's objective is to optimize a linear combination of components constituting the overall business reward defined in Eq. (6), while adhering to linear constraints outlined in Eqs. (2) through (4), along with a nonlinear constraint specified in Eq. (2) that accounts for the total demand lost due to insufficient inventory. We linearize the maximum constraint in Eq. (2) by introducing an additional LP variable. By leveraging perfect information to generate actions, the LP solution gives us with theoretical upper bounds for a given environment.

5 RESULTS AND DISCUSSION

In this section, we compare the performance of DEZ-DQN+GVF with ϵ-greedy strategy as well as heuristic and LP based theoretical upper bound. Our analysis encompasses the training phase results for all algorithms, and we also evaluate and compare their performance on testing data. We evaluate all the algorithms over the business reward that is defined in the problem formulation.

Algorithms Compared: In addition to the baseline algorithms discussed in the preceding section, we extend our analysis to include the following algorithms:

- *DQN:* In our experimental setup, both the general value functions (GVFs) and the primary Q-value function are trained using the DQN framework [23]. Thus, our baseline for comparison is the DQN algorithm. DQN employs ϵ-greedy exploration with a gradually annealing ϵ value, a strategy we also adopt for our experiments.
- *DQN+GVF:* This algorithm leverages GVFs solely for representation learning through a shared representation, which is simultaneously updated by the primary DQN agent and all the GVFs. Similar to DQN, it incorporates ϵ-greedy exploration for action selection.
- *DEZ-DQN+GVF:* In this variant, GVFs not only contribute to improved representation learning but also play a role in action sampling for directed exploration. For a comprehensive understanding of the hyper-parameters used in all these algorithms, please refer to Appendix 1 in the supplementary materials.

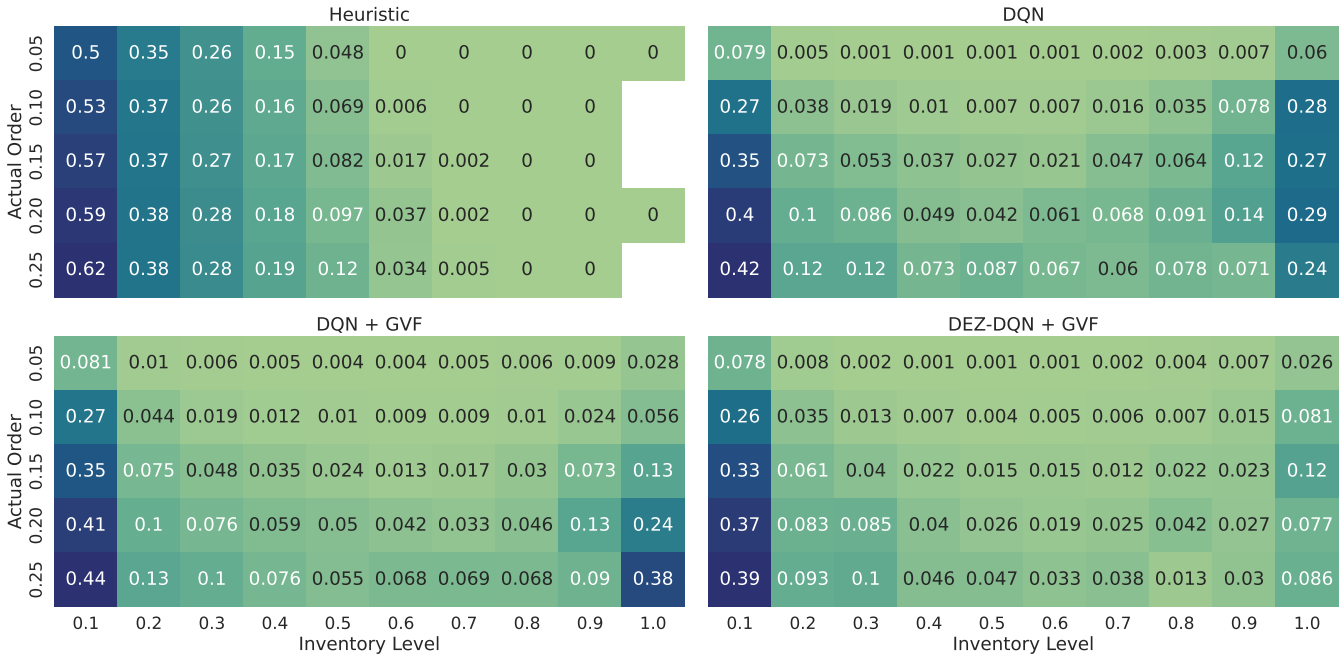

Figure 4: Visualizing main policy heatmaps: A comparative analysis across various algorithms using the 6k product dataset

5.1 Training results

From Figure 3, it is evident that the converged value for the averaged business reward is the same across DQN, DQN+GVF, and DEZ-DQN+GVF on 100, 220, and 6k products datasets. But they significantly outperform the proportional control-based heuristic and we also observe that they are able to achieve 92%, and 86% of the LP-based upper bound for the 100, and 220 products datasets, respectively. The LP solver did not complete its optimization process within the allocated computational time for the 6k products dataset. Additionally, the dual gap is very large at the end of the computational period so LP Solver is not able to find feasible solution for 6k product dataset. Here note that the LP-based upper bound provides us with theoretical upper bounds for a given environment as it uses the perfect information to generate actions. We can also observe that the DEZ-DQN+GVF have better-averaged business reward during the initial exploration and achieve 70%(approx) of converged value faster compared to DQN and DQN+GVF for 100 product dataset. Note that the DQN line is not clearly visible due to overlap with DQN+GVF. For a comprehensive understanding of the performance of all the algorithms, we provide a detailed component-wise analysis of the business reward in Appendix 2.

5.2 Testing Results

In Figure 4, the heatmap depicts the mean replenishment quantity associated with specific inventory and order bins, where, on the x-axis, an inventory level of 0.2 denotes inventory levels ranging from 0.1 to 0.2, and on the y-axis, an order value of 0.10 represents order values between 0.05 and 0.10. The heatmap values are further averaged across five random seeds. (Please note: white space in the heatmap indicate values not available).

Analysis of the heatmaps reveals a clear pattern: when the inventory level remains constant and order values increase, the replenishment quantity also rises. Conversely, when the order value remains fixed and inventory levels are higher, the replenishment quantity tends to be lower compared to situations with lower inventory levels. This consistent trend is observed across all the algorithms. It is worth noting that in all algorithmic heatmaps, there are instances on the right side where higher replenishment appears to occur despite high inventory levels. This happens mostly to due to fact that there is a higher variance in demand in the training data and we are using forecast demand in the state and not the actual demand of the products, so there is chance of going out of stocks if the demand spike is high enough.

Furthermore, the heatmap clearly shows that the DEZ-DQN+GVF agent consistently takes small replenishment actions compared to other algorithms. This behavior can be explained by the fact that DEZ-DQN+GVF explicitly learns the GVF corresponding to the wastage signal and strives to maintain low inventory levels because wastage is directly proportional to inventory levels. This distinction is not observed in the DQN+GVF heatmap, even though it also learns GVFs explicitly from the wastage signal. This disparity may be attributed to the difference in how GVFs are learned in the two algorithms. In DQN+GVF, GVFs are learned from off-policy data (trained on sampled actions from the main policy) and lack corrective feedback from the GVF's policy. In contrast, DEZ-DQN+GVF benefits from GVFs' policies for action sampling during exploration, which results in more efficient control over inventory levels. Furthermore, the heatmap plots corresponding to datasets containing 100 and 220 products can be found in Appendix 3 within the supplementary materials.

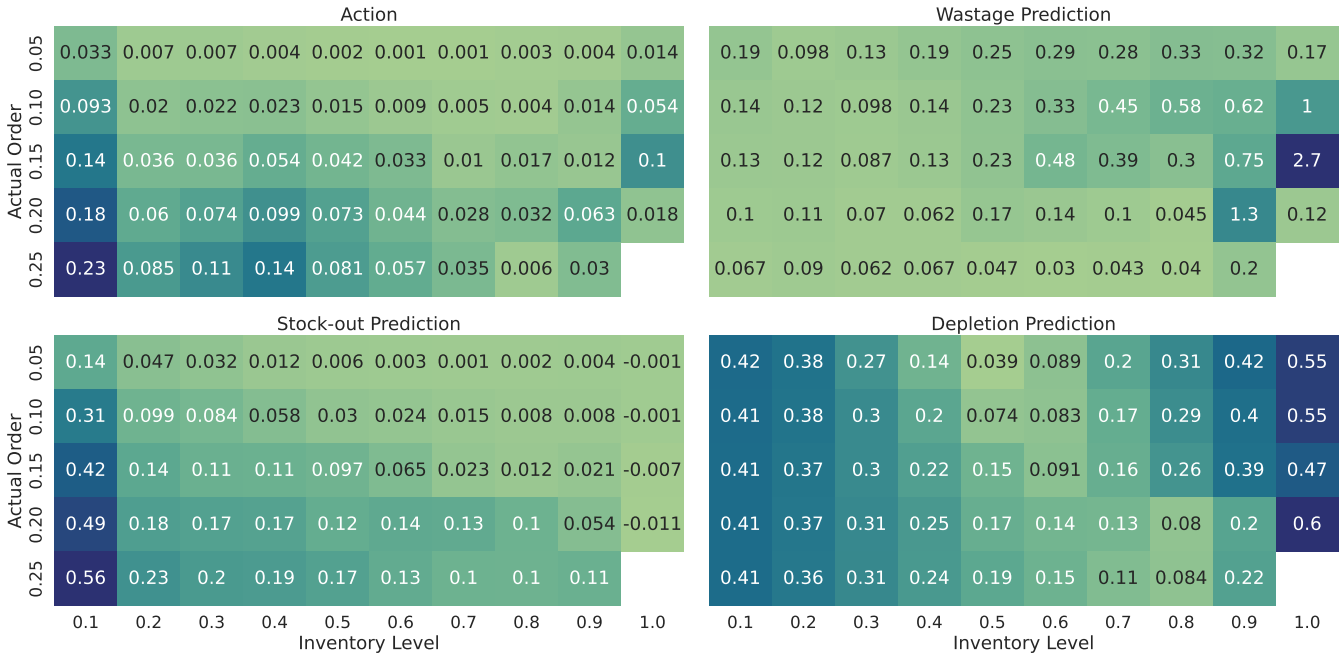

Figure 5: Visualizing averaged replenishment decisions and corresponding GVF predictions for the 220-product dataset: The top-left figure displays a heatmap of the averaged replenishment actions selected by DEZ-DQN+GVF's primary policy, while the other three figures show heatmaps of the averaged GVF quantity predictions (wastage, stock-out and depletion) corresponding to the chosen actions.

5.3 Transfer learning, explainability, and generalisation

Transfer Learning: Table 2 presents the training and testing performance of all algorithms over business reward averaged over 5 random seeds on all datasets. In order to assess the transfer capability of the learnt GVFs to new scenarios, we test all the algorithms on out-of-distribution datasets (ones they are not trained on). It is noteworthy that the datasets encompassing 100 products, 220 products, and 6k products are completely distinct, exhibiting no overlap between them. Our results demonstrate that DQN, DQN+GVF, and DEZ-DQN+GVF consistently outperform the heuristic even when trained on different datasets.

Explainability: Additionally, we can gain insights into the actions chosen by the DEZ-DQN+GVF policy by examining the heatmaps of the GVF predictions. The three learned GVFs play a crucial role in illuminating essential system characteristics, offering predictive cues for selecting appropriate actions. This visualization is presented in the figure 5, which showcases the average replenishment decisions for the dataset of 220 products alongside the predictions of three GVFs (namely wastage, stock-out, and depletion) corresponding to inventory levels and actual demand.

- *Wastage Prediction Heatmap:* Our model links wastage directly to inventory levels and product shelf-life. Consequently, the heatmap reveals that when inventory levels are low, wastage predictions are low, and vice versa. Additionally, when demand is high at a particular inventory level, wastage predictions tend to be low as well.
- *Stock-out Prediction Heatmap:* Clearly, lower inventory levels coupled with high demand yield high stock-out predictions. As inventory levels increase, we observe a smooth transition in stock-out predictions, as the likelihood of running out of stock diminishes.
- *Depletion Prediction Heatmap:* In this case, we aim to understand inventory depletion levels as we make replenishment decisions to maintain desired inventory levels. The heatmap indicates that lower inventory levels result in higher depletion predictions. Interestingly, there are instances where high depletion predictions occur even with high inventory levels. This phenomenon is attributed to the significant demand variability in the training data, which can lead to stock-outs if demand spikes are substantial.

Because these GVF predictions directly relate to tangible system attributes, they offer a clear and interpretable perspective for human supervisors. Consequently, the values of chosen actions from the primary policy can be explained using these GVF predictions.

Generalisability: Finally, the generalisability of the GVF approach can be demonstrated by the ability of the algorithm to adapt to significant deviations in the reward structure. Figure 6 shows the training performance on the 100 product dataset for modified reward definitions. In the figure on the left, agents are fine-tuned on a modified reward in which the weightage on wastage is increased by a factor of 4. In the figure on the right, agents are fine-tuned on the modified reward in which the out-of-stock penalty is given at an inventory level of 0.1 instead of the original 0.05. We do the

Algorithm	Training	Testing		
		Self	Transfer learning	
100 products			Trained on 220 prods	Trained on 6k prods
Heuristic	0.431	0.506	-	-
DQN	0.722	0.723	0.708	0.708
DQN+GVF	0.723	0.727	0.716	0.707
DEZ-DQN+GVF	0.720	0.724	0.717	0.709
LP-upper bound	0.780	0.780	-	-
220 products			Trained on 100 prods	Trained on 6k prods
Heuristic	0.507	0.493	-	-
DQN	0.653	0.649	0.614	0.636
DQN+GVF	0.653	0.654	0.602	0.626
DEZ-DQN+GVF	0.646	0.651	0.620	0.632
LP-upper bound	0.749	0.749	-	-
6k products			Trained on 100 prods	Trained on 220 prods
Heuristic	0.345	0.345	-	-
DQN	0.457	0.573	0.519	0.533
DQN+GVF	0.457	0.536	0.534	0.541
DEZ-DQN+GVF	0.450	0.567	0.535	0.554
LP-upper bound	DNF	DNF	-	-

Table 2: Training and testing performance over business reward averaged over 5 random seeds for all datasets. Note that training and testing results are on 900 and 496 samples respectively, leading to higher average rewards on testing data sets. While the three RL based approaches have similar saturation rewards, their rates of convergence are different.

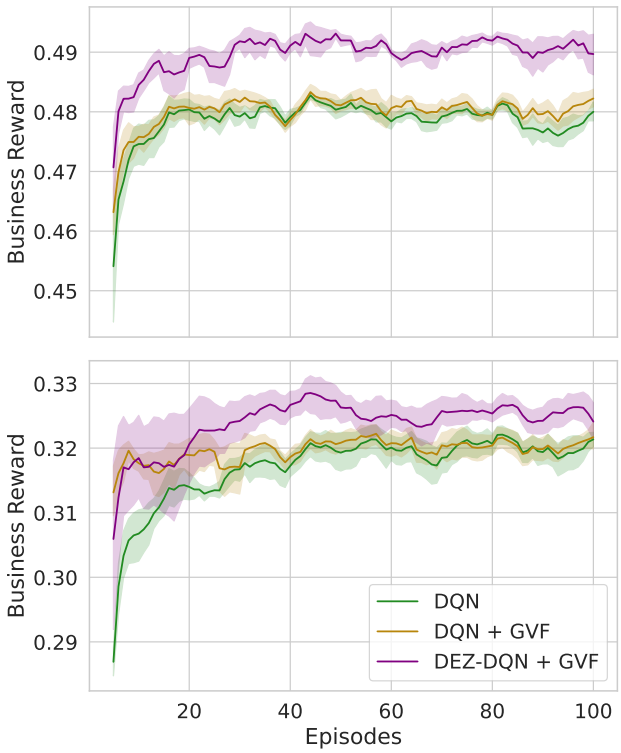

Figure 6: Generalisation on the 100-product dataset with the modified business reward definitions [Note: upper figure - wastage component of the business reward is changed, lower figure - empty level component of the business reward is changed.]. The solid line depicts the mean performance across 5 random seeds, while the shaded region represents a 95% confidence interval.

fine-tuning for 100 episodes with a constant exploration of 0.1, starting from the original trained models. From the figure, it is evident the DEZ-DQN+GVF has a better performance compared to DQN and DQN+GVF, which shows that the DEZ-DQN+GVF can adapt to different business reward definitions. For a comprehensive understanding of the performance of three algorithms, we provide a detailed component-wise analysis of the business reward in Appendix 4.

6 CONCLUSION AND FUTURE WORK

In conclusion, we showed that the use of GVFs can be extended to realistic problems such as inventory management. Using the GVFs gave us better exploration leading to faster convergence, although the saturation reward levels in training were similar. However, the GVFs also gave us the ability to adapt to new tasks within the same environment, such as an abrupt change in the reward structure. In our current work we have hand-designed attributes fed as cumulants for learning the system dynamics, in future we plan to instinctively identify the crucial attributes from the environment and use these cumulants for much better predictive decisions.

REFERENCES

[1] Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.

[2] Souvik Barat, Harshad Khadilkar, Hardik Meisheri, Vinay Kulkarni, Vinita Baniwal, Prashant Kumar, and Monika Gajrani. 2019. Actor based simulation for closed loop control of supply chain using reinforcement learning. In *Proceedings of the 18th international conference on autonomous agents and multiagent systems*. 1802–1804.

[3] Yaman Barlas and Baris Gunduz. 2011. Demand forecasting and sharing strategies to reduce fluctuations and the bullwhip effect in supply chains. *Journal of the Operational Research Society* 62, 3 (2011), 458–473.

[4] Dimitri Bertsekas and John N Tsitsiklis. 1996. *Neuro-dynamic programming*. Athena Scientific.

[5] Dimitris Bertsimas and Aurélie Thiele. 2006. A robust optimization approach to inventory theory. *Operations research* 54, 1 (2006), 150–168.

[6] Real Carbonneau, Kevin Laframboise, and Rustam Vahidov. 2008. Application of machine learning techniques for supply chain demand forecasting. *European journal of operational research* 184, 3 (2008), 1140–1154.

[7] Wenbo Chen and Huixiao Yang. 2019. A heuristic based on quadratic approximation for dual sourcing problem with general lead times and supply capacity uncertainty. *IISE Transactions* 51, 9 (2019), 943–956.

[8] Andrew J Clark and Herbert Scarf. 1960. Optimal policies for a multi-echelon inventory problem. *Management science* 6, 4 (1960), 475–490.

[9] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. 2017. OpenAI Baselines. https://github.com/openai/baselines.

[10] John Doyle, Keith Glover, Pramod Khargonekar, and Bruce Francis. 1988. State-space solutions to standard H2 and H-infinity control problems. In *1988 American Control Conference*. IEEE, 1691–1696.

[11] Richard Ehrhardt. 1984. (s, S) policies for a dynamic inventory model with stochastic lead times. *Operations Research* 32, 1 (1984), 121–132.

[12] Jiarui Fang, Lei Zhao, Jan C Fransoo, and Tom Van Woensel. 2013. Sourcing strategies in supply risk management: An approximate dynamic programming approach. *Computers & operations research* 40, 5 (2013), 1371–1382.

[13] Joren Gijsbrechts, Robert N Boute, Jan A Van Mieghem, and Dennis J Zhang. 2022. Can deep reinforcement learning improve inventory management? Performance on lost sales, dual-sourcing, and multi-echelon problems. *Manufacturing & Service Operations Management* 24, 3 (2022), 1349–1368.

[14] Nir Halman, Diego Klabjan, Mohamed Mostagir, Jim Orlin, and David Simchi-Levi. 2009. A fully polynomial-time approximation scheme for single-item stochastic inventory control with discrete demand. *Mathematics of Operations Research* 34, 3 (2009), 674–685.

[15] Erik Hofmann and Emanuel Rutschmann. 2018. Big data analytics and demand forecasting in supply chains: a conceptual analysis. *The international journal of logistics management* 29, 2 (2018), 739–766.

[16] Nan Jiang and Alekh Agarwal. 2018. Open problem: The dependence of sample complexity lower bounds on planning horizon. In *Conference On Learning Theory*. PMLR, 3395–3398.

[17] Kaggle. Retrieved 08-2018. Instacart Market Basket Analysis Data. https://www.kaggle.com/c/instacart-market-basket-analysis/data.

[18] Durgesh Kalwar, Omkar Shelke, Somjit Nath, Hardik Meisheri, and Harshad Khadilkar. 2022. Follow your Nose: Using General Value Functions for Directed Exploration in Reinforcement Learning. *arXiv preprint arXiv:2203.00874* (2022).

[19] Robert S Kaplan. 1970. A dynamic inventory model with stochastic lead times. *Management science* 16, 7 (1970), 491–507.

[20] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems* 29 (2016).

[21] Harold Joseph Kushner and Dean S Clark. 2012. *Stochastic approximation methods for constrained and unconstrained systems*. Vol. 26. Springer Science & Business Media.

[22] Hardik Meisheri, Nazneen N Sultana, Mayank Baranwal, Vinita Baniwal, Somjit Nath, Satyam Verma, Balaraman Ravindran, and Harshad Khadilkar. 2022. Scalable multi-product inventory control with lead time constraints using reinforcement learning. *Neural Computing and Applications* 34, 3 (2022), 1735–1757.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[24] Steven Nahmias and Stephen A Smith. 1993. Mathematical models of retailer inventory systems: A review. *Perspectives in Operations Management: Essays in Honor of Elwood S. Buffa* (1993), 249–278.

[25] Steven Nahmias and Stephen A Smith. 1994. Optimizing inventory levels in a two-echelon retailer system with partial lost sales. *Management Science* 40, 5 (1994), 582–596.

[26] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. 2006. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX: The 9th international symposium on experimental robotics*. Springer, 363–372.

[27] Jean-Marie Proth, Nathalie Sauer, Yorai Wardi, and XL Xie. 1996. Marking optimization of stochastic timed event graphs using IPA. *Discrete Event Dynamic Systems* 6 (1996), 221–239.

[28] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. 2016. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295* (2016).

[29] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[30] Nazneen N Sultana, Hardik Meisheri, Vinita Baniwal, Somjit Nath, Balaraman Ravindran, and Harshad Khadilkar. 2020. Reinforcement learning for multi-product multi-node inventory management in supply chains. *arXiv preprint arXiv:2006.04037* (2020).

[31] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. 2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. 761–768.