Eindhoven University of Technology

MASTER

Deep Reinforcement Learning for Asymmetric One-Warehouse Multi-Retailer Inventory Management

Kaynov, Illya

Award date:
2020

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 09. Oct. 2026

Department of Mathematics and Computer Science

Data Mining Research Group

Deep Reinforcement Learning for Asymmetric One-Warehouse Multi-Retailer Inventory Management

Master Thesis

Illya Kaynov (1230131)

Supervisors:

Dr. Vlado Menkovski - TU/e

Dr. Albert van Breemen - VBTI

Dr. Willem van Jaarsveld - TU/e

Marijn van Knippenberg, MSc - TU/e

Assessors:

Dr.-Ing. Marwan Hassani - TU/e

Eindhoven, November 2020

Abstract

The one-warehouse multi-retailer (OWMR) is a well-studied inventory management problem, due to its wide application in the industry. In [61] and [20], machine learning methods have been successfully applied to the OWMR system. These papers show that the reinforcement learning agents can outperform the heuristic benchmarks by approximately 10%. However, the approach taken in both papers is restricted to symmetric problems: each retailer has the same cost parameters and demand distribution. The algorithms exploit this to reduce the size of the action space. This work attempts to overcome this limitation by proposing a new approach for learning allocation decisions. Proximal Policy Optimization is used to train a neural network to output exact replenishment quantities to control inventory levels of all stock-points in divergent inventory systems. We compare our approach to similar benchmark instances as employed in [61] and [20]. We show that for symmetric instances, our approach outperforms the baseline heuristics by 8%, and reaches similar performance on asymmetric systems.

Contents

Contents **iii**

1 Introduction **1**

2 Preliminaries **5**

2.1 Related Literature 5

2.2 Reinforcement Learning (RL) 9

2.2.1 Reinforcement Learning in Inventory Management 11

2.3 Deep Reinforcement Learning 12

2.3.1 Artificial Neural Networks 13

2.3.2 Reinforcement Learning and Artificial Neural Networks 14

2.3.3 Deep Learning and Deep Reinforcement Learning in Inventory Management 16

3 Models and Methods **19**

3.1 Problem Formulation 19

3.1.1 Single Echelon Inventory Model 19

3.1.2 Multi Echelon Inventory Model 24

3.2 Methodology 27

3.2.1 Application of Proximal Policy Optimization Algorithm in Inventory Management 27

3.2.2 Baseline Policies 33

3.2.3 Evaluation 36

4 Numerical Experiments **38**

4.1 Single Echelon 38

4.1.1 Backlog 40

4.1.2 Lost Sales 42
4.1.3 Computational Complexity 44
4.2 Multi Echelon 45
4.2.1 Symmetric Systems 47
4.2.2 Asymmetric Systems 49
4.2.3 Summary 52
4.2.4 Influence of GAE on training process 53
4.2.5 Computational Complexity 53

5 Conclusions and Future Research **55**
5.1 Conclusions 55
5.2 Limitations and Future Research 57

Bibliography **59**

A Hyper-parameters **66**

B OWMR Scenario Costs **68**

Chapter 1

Introduction

Supply Chain Management (SCM) is responsible for efficiently delivering goods and services from suppliers to consumers. Production and distribution often involve multiple actors, who together form a Supply Chain Network (SCN). A typical SCN may include multiple stages (echelons) of locations through which the goods move from the supplier to the consumer. The supplier might be a manufacturer of spare parts or raw materials, while the consumer can be a machine at a production facility or a customer at a retail shop. Since SCNs involve multiple locations spread geographically, efficient decisions about positioning goods in time and space are crucial for competitive advantage [36].

Inventory plays a vital role in production and distribution. It is essential for handling supply and demand uncertainties. The overall increase in the demand for goods and the highly competitive markets forces companies to adopt innovative approaches to supply chain management strategies and raises the need for efficient inventory management.

While inventory management is mostly concerned with the overall productivity of inventory systems, inventory optimization focuses on decision-making in the face of uncertainty. There are two competing objectives in inventory optimization, namely low inventory costs and high customer service. On the one hand, inventory optimization aims to reduce inventory-related expenses such as transportation, handling, storage costs, and associated risks of theft, spoilage, and natural disasters. On the other hand, ensuring a sufficient amount of goods is essential for achieving high service levels; surplus inventory helps to shield businesses from market uncertainties and satisfy customer demand. In this regard, it is not surprising that the efficient automation of replenishment decisions is common for many businesses.

This study focuses on one specific stochastic multi-echelon inventory optimization problem, widely known as the one-warehouse multi-retailer (OWMR). This problem is well studied in the literature due to its generality and various applications in the real world. Moreover, it is challenging to optimize due to its underlying combinatorial complexity and the dependencies between SCN locations. As a result, exact solution methods are available only for a handful of systems under restrictive assumptions [44]. Therefore, heuristics and numerical methods are common approaches [15].

[44, 15] provide an overview of the solutions developed over the years. A large number of studies assume a policy class, such as the s-type policy. S-type policies refer to reorder point policies that follow a rule: if the inventory falls below a reorder point, an order is placed to raise the inventory to a specified level. Therefore, in this case, the optimization objective is to find the policy parameters - reorder point and order-up-to level. By assuming the s-type policy, the solution space is restricted. This allows performing an analysis by considering the policy as a part of the system dynamics. Numerous papers provide methods to discover policy parameters or calculate the associated costs [59, 23, 18].

Recent studies (discussed in detail in Chapter 2) take a step away from the classical Operations Research methods and make use of Machine Learning (ML) techniques. Among the success stories of ML approaches, Deep Reinforcement Learning (DRL) stands out as a robust framework that allows for the development of sequential decision-making policies for complex stochastic systems. DRL combines Artificial Neural Networks and Reinforcement Learning to form general-purpose approaches that have advanced the state-of-the-art on a multitude of challenging domains. Some of the most notable examples include playing Atari 2600 games directly from pixels [6], solving the Rubik's cube without prior knowledge [1], performing continuous robotic control [50], and winning against human champions in complex strategic games such as GO and Chess [52], and Dota2 [8].

Inspired by these successes, researchers have applied DRL in sequential decision-making for inventory optimization. The most notable works are: 1) the application of neural-dynamic programming [10] for OWMR [61], and 2) Asynchronous Advantage Actor-Critic (A3C) [37] on dual sourcing, lost sales and OWMR problems [20]. Both papers claim an approximate 10% improvement over s-type heuristics in the multi-echelon stochastic control. These two works [61, 20] serve as an inspiration for our current study.

Despite the excellent results, their methods focus on symmetric OWMR inventory systems, where all retailers share the same parameters, such as costs, demand distributions, and lead

times. Therefore, the problem can be reduced to using neural networks to dynamically specify the heuristic policies for the warehouse and the retailers. While it is a creative way to exploit the system's symmetry, this modeling technique does not scale to heterogeneous retailers. Moreover, the inventory models used in these studies include elements that put the baseline policies (s-type heuristics) at a severe disadvantage. [61, 20] incorporate warehouse emergency shipments, capacity limitations at the locations, and partial lost sales in their inventory models. These elements are known to increase the complexity of the system dynamics and result in s-type policies being pushed further from optimality [15]. Therefore, DRL methods have an unfair advantage over the baseline policies.

In this paper, we apply Proximal Policy Optimization (PPO) [50], a policy gradient RL method, for controlling individual replenishment decisions for all stock-points. In contrast to [61, 20], our method allows for scaling on asymmetric systems, where retailers are allowed to have different costs, lead times, and demand distributions. We consider this point to be the main contribution of our study. The inventory models that we use do not allow emergency shipments from the warehouse. Two options are considered in case of stock-out at the retailer: 1) the consumers are waiting for the product to become available (backlog) or 2) the unsatisfied demand is lost (lost sales). Additionally, infinite capacity at the locations enables the s-type heuristics to form a strong baseline to showcase the competitiveness of our method. It has been established that capacity constraints add considerable complexity to multi-echelon inventory optimization [15]. The optimal policy is multi-dimensional and intractable for practical purposes, even for serial systems [24]. Altogether, these elements help to provide a fair comparison of the DRL optimized policies to the s-type heuristics.

We use a discrete-time simulation to numerically compare the performance of the appropriate s-type heuristics and policies represented by artificial neural networks trained with a Proximal Policy Optimization (PPO) [50] algorithm. First, we show the effectiveness of the method by applying it to a single echelon inventory system and comparing the results with the optimal policy obtained by a closed-form solution. Then, we review the canonical lost sales problem and associated near-optimal heuristics. Finally, we investigate a multi-echelon SCN with a central warehouse distributing goods among multiple symmetric and asymmetric retailers. We assume global information, constant lead times, linear costs, and stochastic demand in all systems.

We show that Proximal Policy Optimization is a suitable candidate for finding control policies in the field of stochastic inventory optimization. The neural networks performance as a policy is

evaluated on many scenarios with varying degrees of complexity. The experiments show that the chosen method reaches the performance of the optimal policies or strong heuristics on a single echelon while outperforming the widely-used heuristics on multi-echelon inventory optimization tasks.

The next chapters are divided in the following manner. First, we provide an overview of the related literature (Chapter 2), including Operations Research, RL, and DRL applications in inventory management. Chapter 3 defines the problem formulation, models and the methodology. The numerical experiment results are presented and analyzed in Chapter 4. This paper concludes with a discussion and recommendations for future research.

Research Questions

This study explores if and how DRL can help create a method for solving asymmetric one-warehouse multi-retailer inventory management systems. Thus, we formulate the following main research question of this thesis:

> *RQ: How can Deep Reinforcement Learning contribute to reducing system-wide operational costs in asymmetric one-warehouse multi-retailer stochastic inventory optimization problems?*

To help answering this complex question we divide it into a number of sub-questions:

> *RQ T.1: How can Deep Reinforcement Learning methods be applied to optimizing the replenishment decisions for single node inventory models?*
>
> *RQ T.2: How can Deep Reinforcement Learning methods be applied to simultaneous control of replenishment decisions on multiple locations?*
>
> *RQ T.3: How well can Deep Reinforcement Learning perform on asymmetric one-warehouse multi-retailer stochastic inventory optimization problems compared to the well-established s-type heuristic policies?*

Chapter 2

Preliminaries

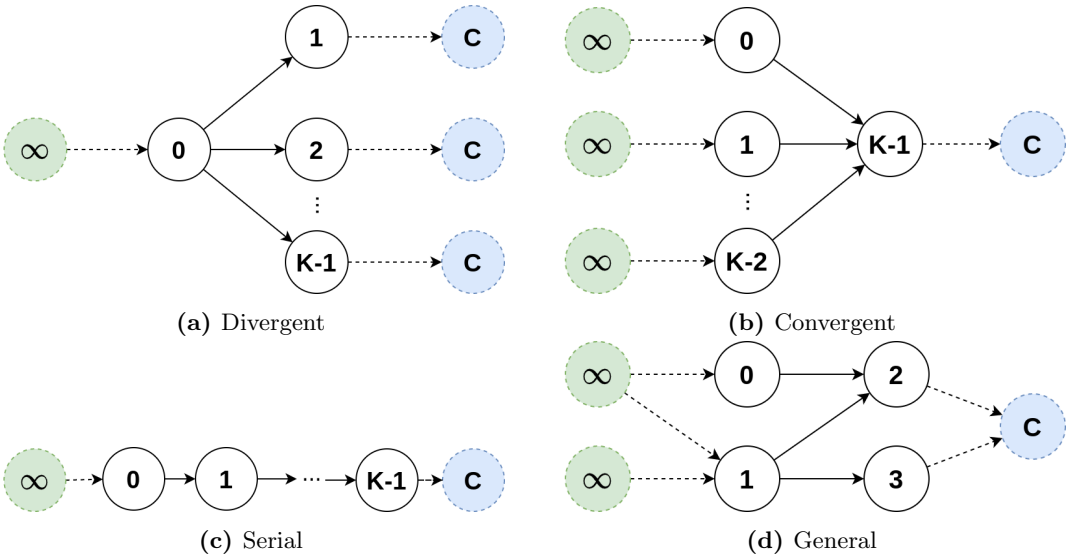
Core concepts and related literature are discussed in this chapter. First, the related literature from an Operations Research point of view is presented, followed by a short introduction to the Reinforcement Learning optimization framework and its corresponding inventory management application. The chapter concludes with an introduction to artificial neural networks and their application to the field of stochastic inventory optimization.

2.1 Related Literature

Supply Chain Management is defined as the collection of processes that control the flow of material goods from the suppliers of the raw materials to the final consumers [3]. Oftentimes, the suppliers, factories, warehouses, and consumers are situated in different locations. Therefore, companies hold stock at multiple geographically dispersed locations to position the goods close to the consumers. The need for thorough control of the number of goods at the locations creates the necessity for inventory management and optimization, which are essential sources of competitive advantage [53].

The locations within a company and external connections can be viewed as a Supply Chain Network (SCN) - an extensive network of organizations that cooperate to fulfill their goals successfully. A typical SCN consists of one or several stages (echelons) that a product needs to pass through from the manufacturer to the end consumer. These are also referred to as single echelon and multi-echelon models. Each echelon can have one or more nodes, where each node represents a location that can hold inventory. In the case of a multi-echelon system, the network structure defines the arrangement of the echelons and the inter-connectivity between the nodes

[15]. Depending on how the actors in the supply chain are connected, SCN can be classified as divergent, convergent, serial, or general. Figure 2.1 provides examples of such systems. Divergent systems are more common in retail and distribution, while convergent systems are more prevalent in manufacturing and assembly industries [15].



(a) Divergent **(b)** Convergent **(c)** Serial **(d)** General

Figure 2.1: Example of different supply chain networks

The literature forms two streams of review policy regarding multi-echelon inventory systems: continuous and periodic review. In continuous review, the goods are monitored in real-time, and the decisions are made as soon as the state of the system changes (i.e., a consumer buys a product). In the periodic review setting, on the other hand, time is divided into decision epochs. The inventory is monitored during the decision epoch, and corresponding replenishment decisions are taken at the end of the epoch, for example, at the end of every day.

Most of the studies in the periodic review stream are based on the seminal work of [13]. The study discusses serial K-echelon inventory models with a single node facing customer demand (where K is the number of locations), illustrated above in Figure 2.1c. The last stock-point $(K-1)$ in the supply chain is facing stochastic demand and can replenish its inventory from its predecessor $(K-2)$. The stock-point $(K-2)$ replenishes its inventory from its predecessor $(K-3)$, and so on. The sequence continues back to stock point (0), which has access to an unlimited supply of goods.

The presented solution is a dynamic programming formulation that uses the *decomposition property* to describe the optimal policy for an N-echelon serial system [13]. The decomposition property is an attribute that allows analyzing the parts of the inventory system in isolation and

form a recursive dynamic programming problem. This method starts by isolating the customer-facing stock-point (K-1) and optimizing its base-stock policy parameters. Next, stock points (K-1) and (K-2) are considered in isolation. It has been shown that the additional cost incurred at stock point (K-1) is a function of the echelon inventory of stock-point (K-2) [13]. Repeating this procedure for all of the previous stock-points allows solving the K-dimensional dynamic program as a single-dimensional recursive DP starting from node (K-1). The final findings indicate that the optimal policy is the echelon-stock policy.

Unfortunately, the decomposition property cannot be applied to divergent systems due to an allocation (rationing) problem. The allocation problem arises when the on-hand inventory at the warehouse is insufficient to satisfy the demand of all downstream locations. The optimal replenishment decisions depend on the warehouse's outstanding orders and the inventory positions of all retailers. Finding an optimal solution, in this case, would involve solving a multi-dimensional dynamic program. This is challenging due to the curse of dimensionality: the exponential growth of computational requirements with respect to the increase in problem size. This, in turn, renders the DP inapplicable to these problems at a realistic scale [16].

One way to circumvent the effect of the allocation problem is to make use of the balance assumption. Making this assumption leads to the decomposition property and the complete characterization of the optimal policy. Furthermore, it relaxes the physical constraints of strictly positive allocation quantities. The balance assumption has various interpretations, such as allowing negative allocation quantities, lateral transshipment between retailers, and permitting the inventory's immediate return to the warehouse with no delay. These interpretations lead to the same result: the retailers' inventory positions become irrelevant, and the allocation decision are based solely on the warehouse echelon stock [16].

The majority of works that consider periodic review divergent systems use the balance assumption in one form or another [16]. Since the optimal policy and the respective costs are unknown, several studies use the relative gap between the system's cost under the balance assumption and the cost of the heuristics (usually obtained through simulation) as a performance measure. The balance assumption leads to the relaxation of the original problem, which produces a lower bound for the real optimal cost. In the case of a small relative gap, the heuristics provide a good approximation of the optimal policy [16].

The effect of balance assumption on the optimal gap has already been extensively studied. The studies report that the balance assumption effectiveness and heuristic quality depend on the

system parameters [16]. The coefficient of variation at the retailers, the lead time at the warehouse, and the difference in the holding costs between the warehouse and the retailers are among the most influential. Despite the balance assumption being widely used to derive close to optimal policies, it is argued not to be suitable when the retailers are prone to imbalance [3]. According to [16], such systems are characterized by different demand distributions at retailers. Precisely these systems are the focus of this study.

Moreover, the vast majority of the studies assume that unmet demand at the last echelon is backlogged. This assumption relates to the consumer's willingness to wait for goods to become available in case of a stock-out. While the backlog assumption is reasonable in some cases (e.g., spare parts management and expensive goods), in others (e.g., in retail with low-cost items and high competition), the lost sales assumption is far more realistic. Nowadays, due to online sales, it has become incredibly easy for the consumer to choose another product or retailer.

Systems under lost sales with positive lead times are challenging to solve [2]. It was established that the optimal policy is not a function of the sum of on-hand inventory and outstanding orders, as it is in the complete back-ordering case [28]. A powerful heuristic policy, the capped base-stock policy, exists for the single-echelon setting with lost sales [66]. However, for multi-echelon systems under lost sales, it is more common to assume instantaneous deliveries between locations to keep the problem tractable [21]. Therefore, the studies for the systems with positive lead times are rare. The capped base-stock policy and its performance in lost sales inventory optimization are further discussed in Chapter 3.

Other approaches for solving stochastic inventory optimization include Sample Average Approximation (SAA) [33]. SAA falls under the Monte Carlo simulation-based approaches and is often used for stochastic discrete optimization problems. The main idea of this method is to estimate the policy cost within certain parameters by performing multiple simulation runs. The results are plugged into deterministic optimization methods to obtain the solution. Due to the fact that numerous sampled scenarios are needed to shape the demand distribution, SAA is also aimed at obtaining parameters of the s-type heuristics.

Additionally, Model Predictive Control and Stochastic Optimization techniques that fall under the umbrella of Mathematical Programming (MP) can also be suited for optimizing stochastic inventory models. However, because of the multi-period and multiple uncertainty realization scenarios, the full formulation of the MP problems can become computationally challenging [35].

We can derive from the literature that there is no general way to solve multi-echelon problems

optimally. The divergent inventory systems are difficult to optimize due to demand uncertainty and allocation decision. Analytical solutions depend on a number of restrictive assumptions and break easily as soon as the assumptions change, or new model components are introduced. Solving the multi-echelon problems sequentially with Dynamic Programming, on the other hand, is not possible due to the curse of dimensionality. Consequently, most studies rely on heuristic policies such as reorder point (s-type) policies, which are known to be sub-optimal in many cases [16]. This leads to the conclusion that the state-of-the-art for intractable inventory optimization problems might be improved by applying data-driven methods such as Approximate Dynamic Programming (ADP).

One of the most promising sets of techniques in ADP is Reinforcement Learning [51]. RL enables the derivation of information-based optimal decision policies, which account for stochastic variations. Model-free RL methods can interactively evaluate sequential policies using only input (system state), output (replenishment decisions), and reward data. The problem is modeled as an environment, and the decision-maker is represented as an agent. Simulation can be used in cases when the underlying statistical probabilities are too complex to capture. The RL forms are a fully data-driven dynamic programming method. RL is better suited for dealing with the curse of dimensionality because it does not require visiting all system states. Moreover, as RL uses a value function to estimate the long-term consequences of actions, it can be applied to control problems that exhibit temporally-extended behavior [57]. In the next section, we provide a quick introduction to the RL optimization framework.

2.2 Reinforcement Learning (RL)

Machine learning can be divided into three main categories: supervised, unsupervised, and Reinforcement Learning. Exemplary inputs and output pairs are available in supervised learning. The algorithm's goal is to generalize on previously unseen examples assuming that the samples are drawn from a stationary distribution. On the other hand, unsupervised learning seeks to find insights into how the data is distributed given only input samples. RL differs from supervised and unsupervised learning due to its interactive nature.

In RL, a problem of sequential decision making is addressed by an agent that learns a mapping from states to actions. The agent accomplishes a task by repeatedly evaluating the effect of its actions on the environment by trial and error [56] (see Figure 2.2). The environment is usually modeled as a Markov Decision Process (MDP). The MDP provides a mathematical framework for

modeling decision-making, which is partially random and partially controlled by a decision-maker.

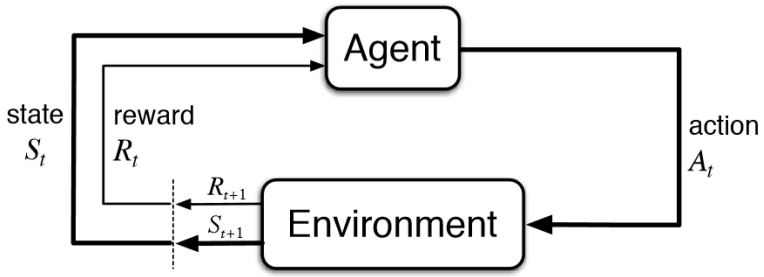


Figure 2.2: Agent environment interaction [56]

A typical MDP consists of a state space, action space, and a transition function. The state-space $\mathcal{S}$ includes all possible configurations $s \in \mathcal{S}$ of the system that can be observed by the agent. The action space $\mathcal{A}$ encompasses all possible actions $a \in \mathcal{A}$ that the agent can choose to influence the system. The transition function $T : \mathcal{S} \times \mathcal{A} \in [0, 1]$ defines a matrix that contains the probability $T(s, a, s')$ given a state s and an action a, the system will evolve to the state s'. The agent can evaluate its actions by a scalar reward signal provided by a reward function $r \in R(s, s', a)$. Such formulation is applied to the tasks where the performance of the agent cannot be evaluated by the means of momentary rewards. Instead, the agent's goal is to learn a mapping from the states to the actions in such a way that the expected cumulative sum of reward is maximized. Therefore, RL addresses the fundamental problem of credit assignment.

The credit assignment problem relates to the process of identifying specific actions that produced the desired outcome. To overcome this challenge, RL makes use of the Bellman optimally equation, that allows expressing the value of the current state in terms of discounted future rewards:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V_\pi(s')] \tag{2.1}$$

The value of the state measures how good a certain state is if we follow the policy $\pi(a|s) = P(\mathcal{A} = a \mid \mathcal{S} = s)$ from this point onward. The discount parameter γ is used to control the contribution of future rewards to the current value calculation. Once the optimal value function is known $V^*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_\pi(s')]$, the optimal policy π^* can be derived with a transition model and Dynamic Programming [9]. However, if the state space is too large or the transition model is unknown, the solution cannot be found in a reasonable amount of time.

The RL algorithms are approximate solutions that tackle these two problems [56]. Unlike DP,

RL avoids the requirements of visiting all the states as it explores only the relevant parts of the environment. However, the exploration/exploitation trade-off is made in this case. Since the policy evaluation is based only on the reward feedback, the agent does not know whether the chosen action was optimal. Thus, the agent must explore alternative actions to increase the probability of finding new, better policies. A balanced trade-off between the use of obtained knowledge (exploitation) and obtaining new knowledge (exploration) is required to achieve maximum reward [56].

2.2.1 Reinforcement Learning in Inventory Management

In multi-period inventory problems, a replenishment decision (action) can influence the total costs of the system for many time-steps into the future. Therefore, an agent needs to consider a sequence of actions that leads to the desired outcome. Additionally, actions can be associated with arbitrary delivery delays. Even if the lead times between the stock-points are constant, they can be viewed as stochastic due to the possible stock-outs at the upstream locations. This problem further complicates the analysis. In the case of multi-echelon divergent systems where the allocation problem occurs, the optimal decisions depend on the full state of the system. Due to the fact that RL is less affected by the curse of dimensionality, we consider it a suitable framework for addressing the problem of decision-making in stochastic inventory optimization. The application of RL in inventory management is not a novelty. This section provides a brief overview of the previous studies performed on this subject.

The work of [19] was one of the first to apply RL to control a serial multi-echelon system. The authors use Semi Markov Average Reward Technique (SMART) [14] to control all stages in a serial system. The algorithm's input is a vector of three variables that correspond to the stock-points' inventory levels. This paper shows that RL can develop near-optimal policies for managing the inventory of serial multi-echelon systems [19].

The SMART algorithm was also used by [45] to optimize the global coordination of distributed supply chains. The problem was represented by multiple manufacturing facilities connected to multi-national retailers. The countries are associated with different markets, currency, and prices. The distance from the manufacturer or distribution centers varies significantly between the nodes. Such a layout requires a high level of coordination to ensure that the system-wide cost is minimized and the revenue is maximized. According to [45], Global Supply Chain Management can benefit from exploiting its distributed structure to achieve a competitive advantage. However, this is usually challenging to achieve with heuristic policies.

Q-learning was applied to the famous Beer Game, developed by the Massachusetts Institute of Technology (MIT) to illustrate the need for coordination and information sharing in the supply chain [62]. The game is a simplified model of the serial multi-echelon inventory system used to showcase the bullwhip effect. The bullwhip effect refers to a phenomenon that small changes in demand at the most downstream node cause huge variability in the upstream nodes' orders. The classical Beer Game consists of four locations connected serially. An agent is assigned to each location to determine how many products to order from its predecessor. Even though the bullwhip effect was not entirely reduced, the study showcases the RL algorithm's potential when applied to inventory management problems [62].

A subsequent work considers the Beer Game with global information: the inventory positions of all supply chain actors are available [12]. The Q-learning algorithm is used to choose one out of 4 distinct order sizes to minimize the system's total cost. The authors compare their results to the genetic algorithm-based solution and showcase the RL approach's superiority in finding replenishment policies [30].

A more recent work of [27] applies SARSA and Q-learning algorithms to manage perishable goods under random demand and deterministic lead times. The authors demonstrate that SARSA or Q-learning applications that include age information into the state achieve superior performance over genetic algorithm-based solutions. Notably, the RL showed better performance when demand variance was high and the goods' life was short.

The studies presented in this section provide evidence that RL is suitable for application in inventory management. However, one of the main downsides of the classical RL algorithms is that they are also obstructed by the curse of dimensionality. As the state and action spaces increase, the computational requirements of the RL methods grow exponentially. Currently, there is no RL solution available to handle the multi-echelon divergent systems considered in this study. The most promising approach to broaden the RL's application field is the use of function approximators, such as Artificial Neural Networks, to estimate optimal value functions. We discuss the Deep Reinforcement Learning approaches in the next section, together with a short introduction to the inner workings of a neural network.

2.3 Deep Reinforcement Learning

The combination of the Reinforcement Learning (RL) framework and Artificial Neural Networks (ANN) defines the field of Deep Reinforcement Learning (DRL). If either actions or states are con-

tinuous or too large, a function approximator is used to estimate the optimal value function $V^*(s)$ and policy $\pi^*(s)$. Neural Networks are the most popular choice for such a function approximator due to their scalability and automated feature extraction. This section presents an introduction to the inner workings of an ANN, followed by an overview of the DRL algorithms.

2.3.1 Artificial Neural Networks

The background work for neural networks mostly originates from the late nineteenth and early twentieth centuries. The neural network structure is vaguely inspired by the human brain. It consists of a set of layers, where multiple neurons represent each layer. Each of the input signals $X = [x_1, x_2, \ldots, x_n]$ is processed by neurons and given a relative weight $[w_1, w_2, \ldots, w_n]$ that determines the impact of input for each neuron.

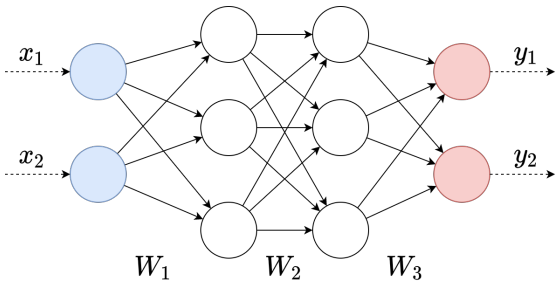


Figure 2.3: Multi-Layer Perceptron architecture. Feed forward neural network consist of input layer (blue circles), a number of hidden layers (white) and an output layer (red)

One of the most common ANN architecture is the Multi-layered Perceptron (MLP), a feed-forward network (presented in Figure 2.3). An MLP consists of a set of parameters θ and activation functions that is used perform non-linear transformation on the input vector X to produce outputs y_i. The layers of MLP can be viewed as a matrix $W_i \in \mathbb{R}^{l_i \times l_{i+1}}$, where l_i is the number of neurons in the layer $i = 1, 2, \ldots, N$ and l_0 equals the number of features in X. Each layer is associated with a non-linear activation function $f_i(x)$, typically a hyperbolic tangent $f_i(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ or a Rectified Linear Unit $f_i(x) = \max(0, x)$. Activation functions allow the network to perform non-linear transformations. Let o_i be the output of the layer i and $o_0 = X$ be the input of the network, then the general expression for a layer i can be written as:

$$o_i = f_i(W_i \cdot o_{i-1} + b_i) \tag{2.2}$$

Each layer transforms the input of the previous layer by performing a matrix multiplication and applying an activation function to the results (Equation 2.2). The task's type influences the

number of neurons and the activation function on the last layer. If the y_i represents the output of the last layer before the activation, softmax $\frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_j}}$ is applied, where K is the number of classification classes. Since the softmax output is normalized and sums up to one, it is suitable for representing a discrete probability distribution over K classes.

In a supervised learning setting, every data point X is associated with a corresponding label. The network weights θ are initialized at random and then adjusted by applying Gradient Descent on a loss function between the network's output and the ground truth. In the case of the classification, the typical loss function is a cross-entropy. The learning rate of α controls the size of the update step. One of the most widely adopted gradient descent algorithms, Adam [31], uses an adaptive learning rate for individual parameters leveraging first and second-order gradient estimates.

2.3.2 Reinforcement Learning and Artificial Neural Networks

This section introduces the concepts, goals, and notation of the Deep Reinforcement Learning framework. DRL represents a family of algorithms that have a goal of estimating the parameters θ of the neural network. The algorithms in DRL can be divided into a number of different ways, for example, value-function based or policy gradient. The value-based method works by directly predicting the value of the state. The policy is derived with acting greedily with respect to the value estimation - picking an action with the highest value. In this case, we would need exploration, which is usually added by taking a random action with a certain probability.

The most popular methods that use Temporal Differencing are DQN [38] and its extensions [60, 64, 47]. These methods assume that a deterministic policy can be the optimal policy. However, initial experiments on single echelon inventory models were characterized by highly unstable training, which suggests the unsuitability of these methods for highly stochastic environments. On the other hand, the policy gradient methods choose an action by sampling it from a probability distribution. The network output consists of two values μ and σ to form a Gaussian distribution for every action dimension in continuous control. Discrete action spaces are addressed by sampling from a discrete probability distribution formed by a softmax activation function. Therefore, Policy gradient methods have trainable parameters that directly affect the stochastic nature of the policy. Thus, they are more suitable for stochastic inventory optimization.

The policy gradient methods are based on the policy gradient theorem. When the policy is differentiable, we can approximate the gradient of the true reward function by taking the gradient

of the policy instead. Let $d^{\pi}(s)$ be a steady-state probability distribution of the Markov Chain under policy π. The gradient of the objective can be written as $J(\theta) = \sum_{s \in \mathcal{S}} d(s) V^{\pi}(s)$. The proof of the policy gradient theorem is too verbose to explore here. The readers can refer to [56]. The theorem shows that the gradient of the true reward is proportional to the gradient of the policy function

$$\nabla_{\theta} J(\theta) \propto \mathbb{E}_{\pi}[Q^{\pi}(s,a) \nabla_{\theta} \ln(\pi_{\theta}(s,a))] \tag{2.3}$$

The most popular algorithm in the policy gradient family is REINFORCE [65], which serves as the basis for virtually all Policy Gradient methods. The REINFORCE training procedure is divided into two stages. First, the trajectories are sampled from the environment and the return $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ is computed for each time-step t. The second part involves the policy update:

$$\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_{\theta} \ln \pi_{\theta}(a,s) \tag{2.4}$$

REINFORCE experiences sample efficiency problems since the trajectories are discarded after an update. The notion of importance weighting allows reusing previous trajectories by calculating the probability ratio between the old and the new policy

$$r(\theta) = \frac{\pi_{\theta}(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} \tag{2.5}$$

At times, the updates might push the network parameters into a part of the parameter space that is hard to recover from. [26] proposes a conservative policy iteration, which restricts the size of the update by providing an explicit bound as a mixture of the old and new policies. [48] presents Trust Region Policy Optimization (TRPO) algorithm that takes the idea further and uses second-order methods to identify the safe update step. Despite the remarkable claims of monotonous policy improvements, TRPO is computationally burdensome. It requires a solution to a constraint minimization problem in every iteration.

The downsides of TRPO are addressed by the work of [50], where Proximal Policy Optimization (PPO) is proposed. The PPO method uses the benefits of TRPO by clipping the policy gradient objective, which allows avoiding the computational burden of the second-order optimization methods. In combination with a value function that provides the advantage estimates, PPO is regarded as a straightforward and robust algorithm that is easy to implement and tune. PPO samples the trajectories in the same way as REINFORCE. However, it is capable of reusing

previous trajectories by means of importance sampling.

All of the previously discussed methods learn from interaction with the environment directly without understanding the tasks' concepts or the underlying dynamics. Such techniques are usually referred to as Model-Free RL algorithms. These algorithms are versatile and straightforward because they do not hold any assumptions about the environment. This, of course, comes at the price of being highly data-intensive, especially when combined with high-capacity function approximators such as neural networks. PPO addresses the sample complexity issue by reusing the collected data with importance sampling. Moreover, PPO allows leveraging multiple CPU cores for the data collection procedure by using multiple workers that interact with separate instances of the environment. Since stochastic multi-echelon inventory optimization requires a large number of scenarios necessary to shape the demand processes [17], it is expected that the policy optimization will require huge quantities of data. Therefore, PPO is considered a suitable algorithm for the task at hand.

2.3.3 Deep Learning and Deep Reinforcement Learning in Inventory Management

The earliest work that utilizes RL with function approximation methods in inventory management is the work of [61]. Neural Dynamic Programming was used to optimize a multi-echelon inventory model. It was able to improve upon the base-stock heuristic by reducing the total costs by 10%. Despite the remarkable results, the method relied on extensive manual feature engineering. For nearly two decades, almost no new studies were published as a successor to this seminal work. Finally, the recent success of DRL algorithms in gaming and robotics has sparked a wave of studies that apply these algorithms in novel and challenging settings.

The work of [40] presents a Deep Q-learning algorithm playing the previously-mentioned Beer Game. The study explores sequential decision-making in a supply chain, where other actors do not follow rational policies. The results show that Deep Q-Networks can develop policies that bring down the total cost of the system by controlling only one stock point. In [40], $d + x$ rule is used to control the inventory levels, where d is the observed demand, and x is the action sampled from a discrete set. The algorithm was allowed to choose out of three possible ordering values for each of the locations.

The work on multi-echelon inventory management was continued by [29]. In their work, they use REINFORCE and SARSA to control the inventory levels of the retailers. However, the

dimensionality of the action space is rather small: only three possible values per actor. Moreover, Radial Basis Function kernels [43] were manually designed for each of the locations to enhance the state representation of the problem. Interestingly, the study uses a demand function that has seasonal variations. However, the baseline was chosen to be a static (s, Q)-policy that is not particularly suitable for a demand distribution with seasonal variations.

The work of [41] provides an interesting take on inventory management. Instead of controlling the inventory levels, the authors use an ANN to predict which node is going to experience a stock-out. Their method is the first to provide robust prediction estimates for multi-echelon inventory models for general network topology. The supervised learning methods were also used in the work of [42], where a single period news-vendor problem is discussed. The demand information is augmented by the features which are used by the Multi-layered Perceptron to unify the demand forecasting and the inventory control into the end-to-end solution. The formulation of this problem in a supervised learning setting is possible due to the goods' perishability: every decision period is treated in isolation.

When the goods are not perishable, however, an action can affect the system's state for many steps in the future. Because of this, [5] apply reinforcement learning for a multi-period news-vendor problem with lost sales. As discussed at the beginning of Chapter 2, the lost sales assumption increases the problem's computational complexity. PPO was successfully applied to derive a replenishment policy in such a case. The authors also showcase that PPO is a suitable algorithm for stochastic optimization by training it on Bin Packing and Vehicle Routing problems. Additionally, the authors raise a question about the DRL experiments' repeatability and propose a benchmark for validating novel solutions.

In [20], the Asynchronous Advantage Actor-Critic (A3C) [37] algorithm is used as a general-purpose technology to find policies for intractable inventory management problems, namely, dual sourcing, lost sales, and OWMR problems. Authors report that A3C developed reasonable policies for all problem settings. However, it struggled to outperform strong heuristics on the lost-sales and dual sourcing problems. Their approach of solving the OWMR problem involved a hybrid between the base-stock and DRL: the output of A3C specified the base-stock level for the warehouse and for all of the retailers at the same time. Therefore, their approach is only suitable for the symmetric systems (identical to the approach of [61]), where all retailers share the same parameters (costs, demand distribution, and lead times).

One of the main problems in applying (Deep) Reinforcement Learning in multi-echelon in-

ventory management is the dimensionality of the state and action spaces. Due to the curse of dimensionality, the convergence speed increases exponentially with the number of variables. Large state spaces are addressed by utilizing neural networks as function approximators [56]. According to the literature, ANNs were proven to be applicable to a variety of domains. In inventory management, the dimensionality of the state space is less of a problem than the dimensionality of the action space. High dimensional actions tend to explode exceptionally quickly. Therefore, most of the works use $d+x$ rule [40], where d is the observed demand, and x is a discrete action, or choose among predefined heuristic policies [20]. In both cases, actions are sampled from a discrete set formed by a Cartesian product of the possible values across each action dimension.

Branching Architecture for Deep Q-Networks was introduced by [58] to address the problem of multi-dimensional action spaces. The Branching Architecture allowed using several action outputs. This architecture was successfully used to address the famous joint replenishment problem [54]. The branching structure reduces the number of discrete actions introduced by several action dimensions from exponential to linear.

However, not only the network architecture needed to be adjusted, but the training algorithm as well. We apply a similar idea but resort to a different class of Deep Reinforcement Algorithms known as Policy Gradient. In comparison with value methods such as DQN, the policy gradients have (1) better convergence properties, (2) are more effective in high-dimensional action spaces, and (3) can learn stochastic policies. In particular, PPO has proven to be a very robust algorithm that is easy to train, tune, and implement. To the best of our knowledge, no approach can control a multi-echelon divergent system with heterogeneous retailers. This current paper fills this research gap. Additionally, PPO was proposed as a method that finds policies for controlling robots with multiple joints. This fact provides the motivation that this algorithm is suitable for simultaneous control of several decision locations.

Chapter 3

Models and Methods

In this chapter, the problem formulation and the formal definition of the inventory models used in this research are presented. Next, the methodology section details the PPO training procedure and defines the heuristic policies and the evaluation method used to compare the performance of the PPO algorithm and the heuristics.

3.1 Problem Formulation

This section describes the inventory models used in this study. We select two supply chain network layouts, namely single-echelon and multi-echelon divergent systems. The single-echelon inventory models with the backlog and lost sales assumptions are presented first in section 3.1.1. We then continue with defining the multi-echelon system in section 3.1.2. Additionally, the Markov Decision Process is formulated for each inventory model.

3.1.1 Single Echelon Inventory Model

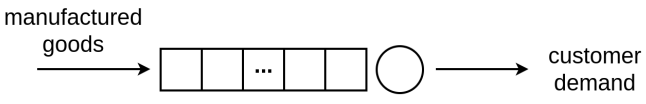


Figure 3.1: Single node system with vendor lead time

A single-echelon system consists of one location (node) that can hold inventory. The location is faced with stochastic demand that is satisfied with its on-hand inventory. The inventory levels are reviewed at regular intervals (periodic review) and can be raised by placing an order at the

manufacturer, which has an infinite supply. However, there is a delay (lead time) between the placement of an order and its arrival at the location. We do not consider cases when the lead time is 0 because the problem is no longer stochastic. When there is no delay, the decision-maker first observes the demand and then places an order. The optimal replenishment size equals the observed demand, and the optimal costs are 0. In our study, the lead time is expressed as a multiple of discrete units of time that without loss of generality, is considered to be one day. The transportation delay and the uncertainty of the future demands create the need for holding inventory.

The squares in Figure 3.1 represents the transportation buffers, and the circle represents the location. At each time-step, an order is placed in the leftmost square. When the system transitions to the next time-step, the goods located in one of these buffers move to the buffer on the right until they reach the final location. The goods can enter or exit the buffers only at the time-steps, which makes the inventory model a dynamic discrete-time system. After a lead time delay l, the goods are delivered at the store and can be used to satisfy the customer demand.

The stochastic demand is realized at every time-step and satisfied with the on-hand inventory. Each unit of demand can be viewed as a customer request for products. In case the on-hand inventory is not enough to fully satisfy the demand, a shortage occurs. Depending on the assumption about customer behavior, one of two cases is considered:

1. The customer is willing to wait for the product to become available. In this case, the model is under a *backlog* assumption. This is suitable for modeling expensive goods or spare parts that are not easy to substitute. The penalty cost p is assigned for every item that the customers are waiting for after the demand is realized.

2. The customer is unwilling to wait and switches to a competitor. In this case the model is under the *lost sales* assumption. The unmet demand is lost, and the penalty cost is calculated in proportion to the size of the unsatisfied order. This assumption is more suitable for retail with low-cost products and highly competitive markets.

Considering all these parameters, the inventory model described above is a dynamic single-echelon multi-period inventory model with vendor lead time and stochastic demand.

Markov Decision Processes Formulation for Single Echelon Inventory Systems

As explained in Chapter 2, Reinforcement Learning (RL) is a framework for finding an approximate solution for large Markov Decision Processes (MDP) for which Dynamic Programming (DP)

methods are intractable. This section presents the definition of MDP for a single echelon inventory optimization problem.

Mathematically, MDP is represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ - state space
- $\mathcal{A}$ - action space
- $\mathcal{T}(S_{t+1}|S_t, A_t)$ - transition dynamic that maps a state-action pair onto a distribution of states at a time $t+1$
- $\mathcal{R}(S_t, A_t, S_{t+1})$ - a reward function that associates a numerical reward with single actions taken from particular states
- $\gamma \in [0, 1]$ - discount factor that controls the influence of future the rewards

The agent observes a state $S_t \in \mathcal{S}$ and outputs an action $A_t \in \mathcal{A}$ that is being fed back into the environment. The environment then changes its state to $S_{t+1} \in \mathcal{S}$ and emits a learning signal in the form of a scalar reward $R_{t+1} = \mathcal{R}(S_t, A_t, S_{t+1})$. This loop continues until the terminal time-step T is reached and the episode terminates. The sequence of states, actions, and rewards constitutes a *rollout* or a *trajectory* of the policy. Every trajectory accumulates rewards from the environment $\mathbf{R} = \Sigma_{t=0}^{T-1} \gamma^t R_{t+1}$.

The objective of the MDP is to discover a policy $\pi : \mathcal{S} \to p(\mathcal{A} = a|\mathcal{S})$ which maximizes the expected cumulative reward $\mathbf{R}$. We can express this objective in terms of finding an optimal policy $\pi^* = \text{argmax}_\pi \, \mathbb{E}[\mathbf{R}|\pi]$ that involves solving the recursive Bellman Equation [7]. Let $\mathcal{S}_t$ represent a set of all admissible states at time step t, then the Bellman Equation can be written in the following form:

$$V(S_t) = \max_{A_t \in \mathcal{A}} \left\{ R_t + \gamma \sum_{s' \in \mathcal{S}_{t+1}} \mathbb{P}(S_{t+1} = s', A_t) * V_{t+1}(s') \right\} \tag{3.1}$$

In the case where a neural network represents a policy, the goal is formulated as finding parameters θ of the policy π_θ that maximizes the expected cumulative sum of rewards:

$$\text{maximize}_\theta \, \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t R_t \right] \tag{3.2}$$

In order to express the inventory model in the MDP terms, first, we have to summarize the notation used to describe the inventory system:

- i - index of stock points, $i = 0$ is the warehouse, and $i = 1, 2, \ldots, N$ are retailers.

- $IL_i(t)$ - inventory level of the stock (on hand inventory - backorders).
- $IL_i^+(t)$ - on hand inventory of the stock point i, where $x^+ = \max(0, x)$
- $IL_i^-(t)$ - number of back orders on the stock point i, where $x^- = -\min(x, 0)$
- l_i - lead time for the stock point i
- $Q_i(t)$ - pipeline vector of the stock point. Items that are expected to arrive at the stock point i. $Q_i(t) = (q_i(t + l_i), q_i(t + l_i - 1), \ldots, q_i(t))$, where $q_i(t)$ indicates the size of an order that is going to arrive at the stock at time-step t.
- $IP_i(t)$ - inventory position of the stock point i, $IP_i(t) = IL_i(t) + \sum_{j=1}^{l_i} q_i(t + j)$. Inventory position indicates the amount of on-hand inventory minus the back-orders plus all of the items that will be delivered to a particular location.
- p_i - penalty cost parameter for a stock point i. The penalty cost parameter regulates the importance of a stock-out. High p usually results in more inventory being stored at the location due to the costly stock-outs.
- h_i - holding cost parameter for a stock point i. Holding costs parameter regulates the cost incurred by the leftover goods after the demand is realized. In this study holding costs are set to one $h_i = 1$ for all problems, while the ratio between holding and penalty costs is controlled by changing the penalty parameter p_i.
- $a_i(t)$ - order placed by the stock point i, which is going to arrive at $t + l_i$.
- $D_i(t)$ - the demand value at time step t incurred at the location i is sampled from a random distribution. The distribution can be normal $\mathcal{N}(\mu, \sigma)$, Poisson $Poiss(\mu)$ or uniform $\mathcal{U}(a, b)$. The values are rounded off to the nearest integer and truncated to 0 to avoid negative values when sampling from a continuous distribution.

As mentioned, the inventory models in this study follow a periodic review policy, meaning that the state of the system is reviewed at regular time intervals. Thus, the systems can naturally be modeled as MDP. Each state transition in MDP is associated with a time-step t and corresponds to a decision period in inventory system. A replenishment decision A_t is made at every time step t, given the current inventory level $IL(t)$, and the vector of outstanding orders $Q(t) = (q(t-l), q(t-l+1), \ldots, q(t))$, where l is the lead time. Since the current inventory model has only one location, indices are omitted, and action A_t is a single integer. The state $S_t = (Q(t), IL(t))$ is a concatenation of the pipeline vector and the inventory position of the location. Thus, the size of the state space is driven by the lead time. An action A_t is assigned on the first position of the pipeline vector $q(t - l) \leftarrow A_t$, where the order will be delivered at the location after the lead time

l. The number of possible replenishment decisions forms the action space. The pipeline vector then evolves by shifting one period forward. Afterward, the orders $q(t)$ are received and added to the on-hand inventory. Figure 3.2 contains an example of the state evolution with arbitrary values.

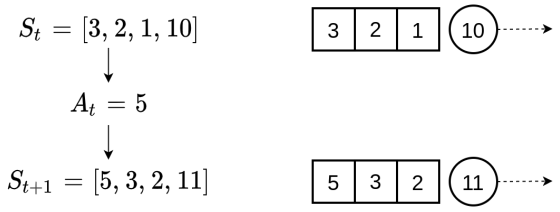


Figure 3.2: The evolution of the state for single node system with exemplary values with lead time $l = 3$ and no demand $D(t + 1) = 0$

Each episode starts with a beginning inventory that equals the mean of the demand μ at the location multiplied by the lead time l, $IL(0) = \mu * l$. Whenever the inventory system is under the backlog assumption, the demand $D(t + 1)$ is realized and subtracted from the on-hand inventory $IL(t+1) = IL(t)+q(t)-D(t+1)$. The excess demand is backlogged, which results in the inventory level falling below 0. In the case of lost sales, the inventory level cannot be lower than 0, thus the state evolves according to the following equation:

$$IL(t + 1) = max(0, IL(t) + q(t) - D(t + 1)) \tag{3.3}$$

The momentary reward is calculated as the negative sum of the holding costs and penalty $P(t)$:

$$R_t = -\left(\sum_{j=0}^{K-1} IL^+(t + 1) * h + P(t)\right) \tag{3.4}$$

For the system with the backlog, the penalty p is assigned for every backlogged item after the demand is realized:

$$P(t) = IL^+(t + 1) * p \tag{3.5}$$

Under the lost sales assumption, the penalty cost is calculated as an opportunity cost per unsold unit:

$$P(t) = max(0, D(t + 1) - IL^+(t)) * p \tag{3.6}$$

Even though the problem formulation (equation 3.1) is relatively simple, the Bellman equation's solution is rather challenging to obtain due to the triple curse of dimensionality. The triple curse

of dimensionality is related to 1) the size of the state space, 2) the size of the action space, and 3) the transition probability matrix. Notably, the curse of dimensionality is relevant only in the case of lost sales. The optimal replenishment decisions are dependant on the inventory level and each element in the pipeline vector. The size of the state space grows exponentially as the lead time becomes longer, thus making it difficult to solve with traditional methods such as Dynamic Programming.

3.1.2 Multi Echelon Inventory Model

A two-echelon inventory model is presented in Figure 3.3. In this setting, random demand materializes at each store (retailer) on every time-step t. The retailers place an order at the upstream location (warehouse), and the orders are filled as much as possible, according to the allocation decision and on-hand warehouse inventory. The warehouse then places an order at the supplier (production decision) with infinite stock. The orders are delivered to the designated locations after a certain transportation delay (from the supplier to the warehouse or from the warehouse to the retailers). This model is widely known as *one-warehouse multi-retailer system* (OWMR).

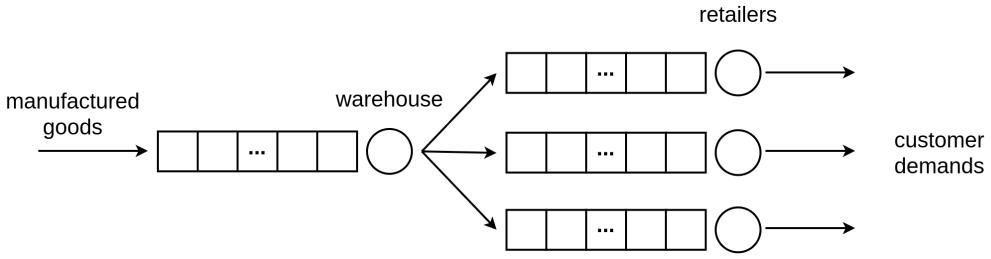


Figure 3.3: Illustration of buffers in one-warehouse multi-retailer systems with lead times

The evolution of the transportation buffers and customer behavior is analogous to the single echelon model described in section 3.1.1. Each retailer can be viewed in isolation as a single node. However, the retailers do not have access to infinite stock, instead, the on-hand warehouse inventory is rationed among multiple locations. There are multiple benefits associated with inventory pooling at the warehouse. One of the main advantages is *risk pooling*. Since trans-shipments between the retailers are not allowed, the surplus inventory at one retailer cannot compensate for a shortage at another. Therefore, pooling inventory at the warehouse can provide the needed redistribution of stock. Additionally, holding inventory at the warehouse is less expensive than holding it at the retailers; for example, the retail stores can be located in shopping centers where storage can be costly. This creates two opposing incentives. On the one hand, the warehouse

should allocate as much to the retailers as possible to reduce potential shortages. On the other hand, positive on-hand inventory at the warehouse can be beneficial if the retailers go out of balance.

The problem formulation is inspired by the works of [61, 20]. Our OWMR model is similar to the ones presented in earlier works, however, we make three modifications, namely:

1. Our model does not consider partial lost sales: 100% of the unmet demand is either backlogged or lost.
2. No emergency shipments from the warehouse are allowed. Emergency shipment refers to shipping goods to the consumer from the warehouse directly bypassing the retailers. The emergency shipment occurs whenever one of the retailers runs out of stock.
3. The locations do not have capacity constraints.

These modifications allow us to formulate a model that increases the competitiveness of the heuristic policies when compared to our method. Additionally, the capacitated locations and emergency shipments negatively affect the quality of the baseline heuristics. Thus, our problem formulation presents a relatively simple inventory model where the use of s-type heuristics is justified.

Markov Decision Processes Formulation for Multi-Echelon Inventory Systems

This part extends the definition of MDP presented for the single echelon model in section 3.1.1 to the OWMR system. The system dynamics and notations are analogous, and any differences are further explained below.

A unique identifier is assigned to every location. The warehouse has index 0, while the retailers are identified with numbers from 1 to $K-1$, where $K \in \mathbb{N}$ is the total number of locations. The inventory flow in the OWMR system starts with the central warehouse raising its inventory level $IL_0(t)$ by placing an order $a_0(t)$ at the manufacturer with an unlimited supply. There is a strictly positive replenishment lead time at the warehouse l_0 and an associated vector of outstanding orders $Q_0(t) = (q_0(t-l_0), \ldots, q_0(t))$. The inventory levels and the pipeline vectors of the retailers are denoted in a similar manner $IL_j(t)$ and $Q_j(t) = (q_j(t-l_j), \ldots, q_j(t))$, for $j \in 1, \ldots, K-1$. The replenishment decisions for the retailers $a_j(t)$ are subject to the following constraints:

$$0 \leq a_0(t), \forall i \in 0, \ldots, K-1 \tag{3.7}$$

$$\sum_{j=1}^{K} a_j(t) \leq IL_0^+(t) \tag{3.8}$$

The state of the MDP $S_t = (Q_0(t), IL_0(t), Q_1(t), IL_0(t), \ldots, Q_{K-1}(t), IL_{K-1}(t))$ is a concatenation of the pipeline vectors and the inventory levels of all the stock-points. Thus, the size of the state vector is influenced by the lead time at all locations $|S_t| = \sum_{i=0}^{K-1} l_i + K$. Furthermore, the action $A_t = [a_0(t), a_1(t), \ldots, a_{K-1}(t)]$ is a vector that combines production and allocation decision for the time-step t. Therefore, the size of the action vector equals the number of locations $|A_t| = K$.

The inventory level of the warehouse is updated according to the following equation:

$$IL_0(t+1) = IL_0(t) + q_0(t) - \sum_{j=1}^{K-1} a_j(t) \tag{3.9}$$

Under the backlog assumption, the inventory levels of the retailers are updated with the following formula:

$$IL_j(t+1) = IL_j(t) + q_j(t) - D_j(t+1) \tag{3.10}$$

In the case of lost sales, the retailers' inventory levels cannot be negative. Therefore a different formula is applied:

$$IL_j(t+1) = \max(0, IL_j(t) + q_j(t) - D_j(t+1)) \tag{3.11}$$

In order to fully characterize the system dynamics, the sequence of the events is specified bellow:

1. Inventory levels are observed, and current production and allocation decisions are determined
2. Warehouse updates its level according to equation 3.9
3. Retailers update their levels according to equations 3.10 or 3.11
4. Reward is calculated according to equation 3.12

A momentary reward is calculated as the negative of all the costs incurred during the time-step t at all locations

$$R_t = -\left(\sum_{i=0}^{K-1} IL_j^+(t+1) * h_j + P_j(t)\right) \tag{3.12}$$

The costs are calculated similarly to the single echelon system: each retailer incurs holding and penalty costs, while the warehouse has only holding costs (i.e. $p_0 = 0$).

MDP's objective in a multi-echelon setting is analogous to the formulation presented in 3.1.1: find an optimal policy that maximizes the expected sum of discounted rewards. The curse of dimensionality, in this case, is even more pronounced than for the single echelon. The state-space depends on the number of locations and their corresponding lead times. The action space is multi-dimensional, where each dimension corresponds to a location. Finally, the transition function is influenced by independent random demand at multiple retailers. Therefore, finding a policy for such a large MDP with multi-dimensional actions is a challenge even for the most sophisticated machine learning techniques. In the next section, we formulate a solution methodology to tackle this problem.

3.2 Methodology

This section presents the methods used in training the neural network as a policy for controlling the replenishment decisions in the OWMR systems. The Proximal Policy Optimization (PPO) algorithm is introduced first, followed by a detailed explanation of its application to stochastic inventory optimization. Finally, the heuristic policies which serve as the baseline for the evaluation are discussed. The section concludes with a description of the evaluation method used to compare the performance of the DRL-derived solutions and the heuristic baselines.

3.2.1 Application of Proximal Policy Optimization Algorithm in Inventory Management

This section describes the specifics of the application of PPO in OWMR inventory optimization problems.

PPO [50] is a policy gradient method that enhances the well-known REINFORCE algorithm [65]. It does so by incorporating elements from the Trust Region Policy Optimization (TRPO), importance sampling, and value methods. TRPO ensures a monotonic policy improvement by leveraging second-order optimization methods [48]. PPO achieves similar results by clipping the optimization objective. Unlike REINFORCE, however, the training data can be reused multiple times by the PPO due to the importance of sampling. In combination with a value function used to improve the training stability, PPO is characterized by stable training with good wall-time performance.

[20] reports that an expensive hyper-parameter tuning process is required for achieving good

performance on inventory optimization problems. PPO has empirically shown to be less sensitive towards hyper-parameter tuning. Additionally, we chose the PPO method to perform stochastic inventory optimization tasks due to its good convergence properties and good sample complexity. An efficient implementation is available from Rllib [34] reinforcement learning library, which is used in this work.

In order to provide individual control over the replenishment decision in OWMR systems, we defined an MDP with multi-dimensional action space in Section 3.1.2. This formulation allows us to solve asymmetric problem instances, where retailers' parameters (lead time, demand distribution) might differ. Moreover, the actions produced by a policy are subject to a constraint 3.8. We elaborate on the architecture of the policy network in the next section and define its interaction with the environment.

Actor-Network and Interaction with the Environment

As discussed in Chapter 2, the PPO algorithm is an actor-critic method that combines policy optimization and value approximation. The actor-network (also referred to as a policy network) $\pi(S_t|\theta)$ is responsible for outputting actions. The actor-network is a Multi-layer Perceptron with two hidden layers. The dimension of the input layer is matched with the number of dimensions of the state space $\mathcal{S}$. The dimension of the last layer is adjusted to the maximum replenishment quantity for each stock-point.

In the case of a single echelon inventory model, we identify 20 integer actions in the range [0, 20). The output layer of the actor-network is being processed by a softmax activation function since there are no constraints on the size of the replenishment orders. The corresponding action is sampled from the final discrete probability distribution. In contrast, the actions in the OWMR system are subject to the constraint specified in equation 3.8. In the literature, this issue is solved by forming a Cartesian product of all possible actions across the dimensions and masking the infeasible actions. This approach results in the exponential growth of the action space as the number of action dimensions becomes larger. We adopt a different approach to this problem, which is discussed next.

The warehouse can choose from 20 integer values in a [0, 20) range, while each retailer has 10 options in the [0, 10) range. The output of the actor-network equals the sum of all possible actions for each location, namely $20 + 10 * (K - 1)$ values. The softmax activation function is applied with respect to the specified action ranges first 20 elements of the output vector form a

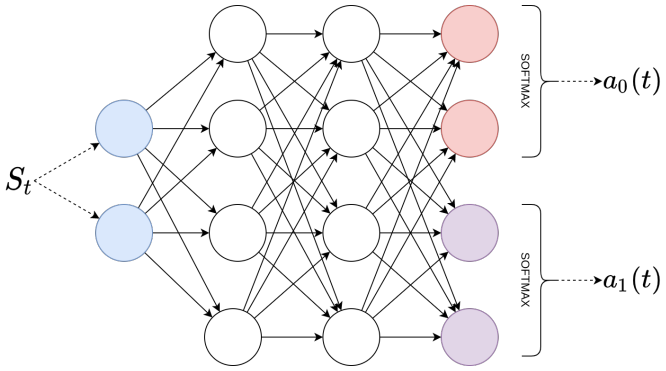


Figure 3.4: Simplified visualization of the actor-network. Dashed arrows represent inputs and outputs, while solid arrows symbolize weights (biases and hidden layers' activation functions are omitted for clarity).

probability distribution over the warehouse's actions, while the rest are divided between the $K-1$ retailers. The final output of the network is a vector $A_t = [a_0(t), a_1(t), \dots a_{K-1}(t)]$, where K is the total number of locations. This allows us to specify an exact ordering decision for each of the stock-points and keep the size of the output layer grow linearly with respect to the number of actions. See Figure 3.4 for simplified visualization of the network.

Even though this method allows us to scale the network's output linearly with the number of locations, it is difficult to restrict the output to the allocation constraint highlighted in Equation 3.8. It is not possible to execute these actions directly when the allocation constraint is violated. This issue raises two questions: 1) how to adjust the actions to become feasible and 2) how to communicate to the RL agent that these actions were not feasible. One of the solutions could be to apply an *allocation rule* that takes the number of available on-hand inventory at the warehouse and suggested actions $A_t = [a_0(t), a_1(t), \dots a_{K-1}(t)]$ and outputs a feasible action $A'_t = [a_0(t), a'_1(t), \dots a'_{K-1}(t)]$. The allocation rule can be proportional. Each proposed action is scaled down according to its size $a_i(t), i \in \{1, \dots, K-1\}$, the available on-hand inventory at the warehouse $IL_0^+(t)$ and the sum of suggested actions $\sum_{j=1}^{K-1} a_i(t)$:

$$a'_i(t) = \left\lfloor a_i(t) * \frac{IL_0^+(t)}{\sum_{j=1}^{K-1} a_i(t)} \right\rfloor \tag{3.13}$$

This rule is favorable for the s-type heuristics as it fairly distributes the available inventory among the retailers with respect to their inventory positions. However, as empirically validated, this proportional allocation does not provide an incentive for the RL agent to output feasible actions. The agent tends to suggest high order quantities at the retailers (in the top of the available range), while limiting on-hand inventory at the warehouse. This results in undesired

behavior: the agent does not learn to ration the resources, instead all inventory is being equally split between the retailers. Even though this approach is not inherently wrong, this leads to poor performance. The problem persists event when the penalty for the infeasible actions is added. This could be related to the fact that rewards are highly stochastic, which means it is challenging for the agent to evaluate which actions are considered infeasible. Additionally, a valid penalty is difficult to define for each problem settings.

While training the RL agent, we rely on the *sequential allocation* rule. We randomize the sequence of the retailers at every time-step, and the suggested (probably infeasible) actions $a_j(t), j \in 1, \ldots, K-1$ provided by the actor-network are executed one by one. The on-hand warehouse inventory is being decreased by the corresponding amount every time an action is executed. If there are not enough goods to execute a suggested action, the action is truncated so that the on-hand inventory at the warehouse does not fall below 0. This means that in case of an infeasible action (violating the constraint 3.8), the last retailer(s) in the sequence will receive a lower quantity than suggested, which will increase the retailer's costs. During the training, the RL agent learns to output feasible actions since the infeasible actions lead to unfavorable costs. Not only this allows us to remove the extraneous penalty for infeasible actions, but it also results in better performance. The performance improvements are maintained even when switching to the proportional allocation rule because the RL agent has learned to use the warehouse resources in the most efficient way. It is important to note that the actions' adjustments are only done inside the environment and do not affect the actions collected in the training batch for the PPO training.

Critic-Network and Policy Improvement

The training process of PPO can be divided into two parts: a collection of experiences and policy improvement. First, the trajectories are generated by the agent-environment interaction and saved into a training batch. The experience collection process can be accelerated by increasing the number of workers distributed among several CPU threads. We use the current policy of the actor to generate a training batch for each training iteration. The sample points in the training batch include observed states, actions, and corresponding rewards. Once the training batch is complete, the value estimates $\hat{V}_t$, the advantages $\hat{\mathbf{A}}_t$, and the true value targets V_t^{target} are calculated in the post-processing step. A policy improvement involves adjusting the weights and biases of the neural nets through adaptive mini-batch gradient descent, using Adam optimizer [31]. The training batch can be reused across multiple epochs.

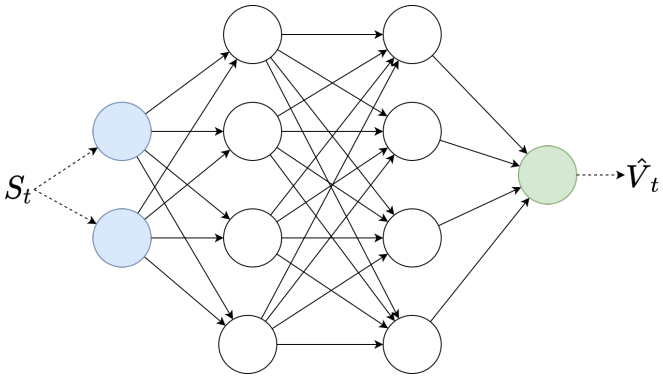


Figure 3.5: Simplified visualization of the critic-network. Dashed arrows represent inputs and outputs, while solid arrows symbolize weights (biases and hidden layers' activation functions are omitted for clarity).

The critic-network is represented by a separate neural network $V_\pi(S_t|\theta')$, which estimates the value $\hat{V}_t$ of the state S_t when following the policy π.

$$\hat{V}_t = V_\pi(S_t|\theta') \tag{3.14}$$

The critic-network structure is identical to the actor-network, but the last layer has only one neuron with a linear activation. The critic-network is visualized in Figure 3.5.

The value estimates are used to calculate the advantage $\hat{\mathbf{A}}_t$ of the action A_t in the state S_t. The advantage indicates how much better the value of an action is with respect to the critic's estimate. The Generalized Advantage Estimation (GAE) is used for calculating the advantage function:

$$\hat{\mathbf{A}}_t = \sum_{l=0}^{T} (\gamma\lambda)^l \delta_{t+l}^V \tag{3.15}$$

where $\delta_t^V = R_t + \gamma\hat{V}_{t+1} - \hat{V}_t$ is the temporal difference residual [49]. The hyper-parameter λ signifies the trade-off between bias and variance of the advantage estimates. High λ values reduce the variance but increase the bias. In a highly stochastic environment, as one presented in this study, GAE allows providing more stable advantage estimates, thus improving training stability. Finally, the value targets are computed with the real rewards recorded in the training batch

$$V_t^{\text{target}} = \sum_{t'=t}^{T} \gamma^{t'} R_{t'} \tag{3.16}$$

Since the actor and the critic-networks are separate networks, we compute two loss functions to update the network parameters. The loss of the value-network is the expected mean squared

error between the discounted rewards V_t^{target} and the value function approximation $\hat{V}_t$ across a training batch:

$$\text{Value Loss} = c_1 \hat{\mathbb{E}}_t \left[(\hat{V}_t - V_t^{\text{target}})^2 \right] \tag{3.17}$$

The term c_1 is a value function loss coefficient. It is used to control the strength of the final value-network loss. The neural network is initialized with random weights, essentially forming a random policy. The coefficient c_1 is used to compensate for the large difference in value estimates and the true value target at the beginning of the training.

The policy loss equals the clipped surrogate objective:

$$J^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{\mathbf{A}}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{\mathbf{A}}_t \right) \right] \tag{3.18}$$

The clipped objective includes an additional hyper-parameter ϵ, which defines the maximum size of the actor-network update. We use the value of $\epsilon = 0.2$ as suggested in the original paper [50].

$$r_t(\theta) = \frac{\pi_\theta(A_t \mid S_t)}{\pi_{\theta_{\text{old}}}(A_t \mid S_t)} \tag{3.19}$$

Equation 3.19 refers to the probability ratio between the old and the new policies. It is based on the idea of importance sampling - the general statistical technique for estimating the properties of one distribution while only having samples generated by a different distribution. The equations 3.19 suggests that the ratio becomes smaller as the difference between the old and the new policy increases. This method allows us to reuse the previously collected experiences multiple times and reduce the magnitude of the policy update if the trajectories become too stale. Importance sampling in combination with the clipped objective $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ results in stable and sample efficient training. By performing gradient ascent on the objective (equation 3.18) the actor-network weights are adjusted to increase the probability of good actions (advantage is positive) and decrease the probability of bad actions (advantage is negative).

The total loss of the actor-network $\pi(S_t|\theta)$ is formed by adding together the clipped surrogate objective and the entropy bonus:

$$J^{CLIP'}(\theta) = J^{CLIP}(\theta) + \hat{\mathbb{E}}_t \left[c_2 H(S_t, \pi_\theta(.)) \right] \tag{3.20}$$

The entropy bonus is related to the exploration/exploitation dilemma in RL. In order to encour-

age exploration, the entropy $H(S_t, \pi_\theta(.)) = \sum_{a \in \mathcal{A}} \pi(a, S_t) \log \pi(a, S_t))$ of the policy is calculated and added to the final objective. If the policy has high entropy, it explores more. On the other hand, the entropy decreases when one action is more probable than the other. The coefficient c_2 controls the contribution of the entropy to the total objective. Notably, a high c_2 value results in a policy that tends to explore more.

Table 3.1 presents the list of the final hyper-parameters that were used for the PPO training. In our case, the PPO algorithm did not require extensive hyper-parameter tuning. Nevertheless, some limited tuning has taken place to ensure that the best possible parameters were chosen. The explanation of the hyper-parameters, as well as the most valuable findings of the tuning process, are detailed in Appendix A.

Hyper-paramer	Value
Number of Layers	2
Number of Neurons	[64, 64]
Activation Function Policy Network	tanh
Activation Function Value Network	ReLU
Train Batch Size	327680
Mini-batch Size	32768
Number SGD Epochs	30
Number of workers	12
Learning Rate	10^{-4}
VF Loss Coefficient (c_1)	10^{-7}
Entropy Coefficient (c_2)	0
Gradient Clipping Norm	40.0
Discount factor γ	0.95
GAE λ	0.98

Table 3.1: Hyper-parameters used for PPO training

3.2.2 Baseline Policies

Single Echelon with Backlog

The base-stock policy is the optimal policy in a single-node system under the backlog assumption. The optimal base-stock level z^* can be determined analytically only if the demand distribution is known. Let $F_l(.)$ be a cumulative distribution function of the l period demand, then the optimal base-stock level z^* can be calculated by first identifying the critical ratio $CR = \frac{p}{p+h}$ and then finding the inverse of the cumulative distribution with respect to CR:

$$z^* = F_l^{-1}(CR) \tag{3.21}$$

The replenishment quantity at each time-step t can be consequently derived by subtracting the inventory position from the base-stock level:

$$a(t) = \max(0, z^* - IP(t)) \tag{3.22}$$

This equation follows from the queue theory. The base-stock levels are set in a way that the probability of the consumer's demand being fully satisfied equals the critical ratio.

Single Echelon with Lost Sales

Lost sales assumption in a single-node inventory system results in higher computational complexity. The optimal policy is dependant on the inventory position and the distribution of the values in the pipeline vector. In the case of lost sales, it is common to use the same s-type heuristic derived from the backlog analog: the base-stock policy. However, the performance of the base-stock policy only approaches optimal in the case of short lead times and high penalty values.

Another class of heuristics, the constant order policies, can be applied for such a system. The constant order policies that place the same replenishment order every time step are proven to be asymptotically optimal as the lead time increases [67]. However, with a decrease in the lead time, the performance starts to deteriorate. [66] proposes a new type of policy that combines the best of the two heuristics into a hybrid *capped base-stock*. The capped base-stock policy is associated with two parameters: a base-stock level z and an order cap r. Better performance is related to the order cap r that results in a smoother policy which clips the values that are larger than r. The capped base-stock behaves as a base-stock when the lead time is small (or the penalty cost is high) and converges to the constant order policy as the lead time increases [66]. Despite its good performance, the cost function is not convex compared to the base-stock level policy [66].

The presence of the order cap results in order-smoothing and constrains high future holding costs. In other words, unexpected large demand might result in an equivalently large order under the base-stock policy, which, in turn, drives the future holding costs up. The effect of the order-smoothing is demonstrated in more detail in [55]. It is expected that DRL will produce a more sophisticated policy structure that allows for a more suitable order-smoothing in complex environments under the lost sales assumption.

Another method for solving lost sales inventory models is tested in [20], namely Linear Programming Approximate Dynamic Programming (LP-ADP). The capped base-stock performance in the experiments of [20] were as good as LD-ADP, while the constant policy performed worse

than the base-stock in all scenarios. We exclude the LD-ADP and the constant policy from our study and only consider the capped base-stock and base-stock policies as the baselines in our experiments.

Multi Echelon

The optimal guarantees of the s-type policy do not carry over from a single echelon to multi-echelon divergent systems. The structure of the optimal policy is unknown and expected to be complex. An optimal decision to transfer goods from one site to another may depend on all status of all sites. Since Dynamic Programming is intractable due to the allocation problem, it is common to use somewhat simpler reorder point policies [3].

There are two commonly used heuristics in this case. First is the decentralized base-stock heuristic policy where the replenishment decisions are only based on local information. In some papers, the base-stock policy in a multi-stage system is referred to as an installation-stock policy. The benefit of this policy is that only local inventory positions are needed. However, the cost effectiveness of the base-stock heuristic is also limited by the lack of information about the entire system. A relatively simple way to incorporate this information is to base decisions on the echelon-stock policy, the second common heuristic we consider. The echelon-stock is obtained by summing up the inventory position at the location with the inventory positions of all downstream locations. In other words, the echelon-stock heuristic results in a policy that orders at the warehouse as soon as the demand is realized at the retailers. This means that this policy has an advantage by being able to anticipate the retailers' orders. [4] has shown that an equivalent echelon-stock policy can replace the base-stock policy.

The formalization of the base-stock and echelon-stock heuristic policies is presented below. The replenishment decisions for the retailers are taken with respect to their base-stock levels z_j identified by an exhaustive search over the parameter space. The base-stock policy replenishment decisions at all locations are derived according to the following equation:

$$a_j(t) = \max(0, z_j - IP_j(t)) \tag{3.23}$$

When the system is being controlled by the echelon-stock heuristic, the warehouse sums its inventory position with the inventory positions of the retailers:

$$a_0(t) = \max(0, z_0 - \sum_{i=0}^{K-1} IP_i(t)) \tag{3.24}$$

Finally, the *allocation rule* should be specified to ensure the feasibility of the replenishment decisions (according to constraint 3.8). We utilize the proportional allocation rule. The actions provided by the heuristics $a_j(t)$ for each retailer $j \in 1, \ldots, K-1$, are truncated according to the following formula:

$$a'_i(t) = \left\lfloor a_i(t) * \frac{IL_0^+(t)}{\sum_{j=1}^{K-1} a_i(t)} \right\rfloor \tag{3.25}$$

We use the base-stock heuristic as a baseline in our comparison, following the example of [61, 20]. Additionally, we add an echelon-stock heuristic, due to its strong performance and suitability in the selected OWMR problems.

3.2.3 Evaluation

A discrete time simulation is used to calculate the total costs of different policies. We modeled the heuristic and reinforcement learning policies as agents that interact with simulation through Open AI Gym interface [11]. We use the proportional allocation rule for the base-stock, the echelon-stock and the PPO agents when calculating the costs for the OWMR systems.

We adopt a cost-based approach for evaluating the heuristics and the neural network-based policies trained by PPO. The rewards are expressed as negative costs of the inventory system. Therefore, we compare the total savings attained by each agent using the average sum of undiscounted rewards:

$$\bar{R} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} R_t \tag{3.26}$$

The results presented in the next chapter are produced by running $N = 1000$ episodes with $T = 100$ time-steps (decision periods) each. We calculate the average undiscounted rewards across those trials together with a standard error:

$$\sigma_{\bar{R}} = \frac{\sigma}{\sqrt{N}} \tag{3.27}$$

where σ is the standard deviation.

We proceed with normalizing the results with respect to the heuristic policies and express the costs difference in percentages. This method allows us to compare the different problem instances. Additionally, during the evaluation, we use the same random seed for N runs for all agents to ensure fair comparison. The agents are evaluated on the same sequence of demand values sampled from a random probability distribution. We identify the best parameters for all heuristics by

performing an exhaustive search over the parameter space based on average sum of rewards $\bar{R}$.

Chapter 4

Numerical Experiments

The main objective of this section is to present and analyze the results of the numerical experiments we have conducted based on the methodology presented in Chapter 3. We then discuss the suitability of our methodology for the stochastic inventory optimization tasks.

We first experiment with the smaller and simpler single-echelon systems to gain insight into the effectiveness of our method, so we can later apply it to the more complicated multi-echelon setting. After the single echelon, this chapter concludes with experiments on multi-echelon one-warehouse multi-retailer (OWMR) systems. We divide the problems into symmetric and asymmetric cases regarding the lead times and demand distributions of the retailers. We also consider the backlog and the lost sales assumptions.

4.1 Single Echelon

As mentioned earlier in this work, the optimal policy for a single-echelon model with backlog is the base-stock policy. The optimal parameters are calculated with a closed-form solution, as shown in Chapter 3. Therefore, the single-echelon model (Figure 4.1) serves as a benchmark to test the methodology against the optimal solution.

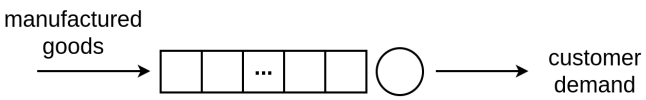


Figure 4.1: Single node system with vendor lead time

The study of simple single-echelon systems provides valuable insight into the behavior of heuristics and RL agents in a multi-echelon divergent SCN. It helps to evaluate how well PPO can

approximate optimal and asymptotically optimal policies. Moreover, the single-echelon models under the lost sales assumption form classical intractable inventory optimization problems for which the solution cannot be easily obtained.

In order to cover a number of possible scenarios within a business context, we generate several problem settings considering two dimensions: lead time and penalty costs. The set of parameters for the lead time l and penalty costs p are chosen from the Cartesian product of the two discrete sets $l = \{1, 2, 5\}$ and $p = \{1, 9, 27\}$. The values for the lead time were chosen to cover the short $l = 1$, medium $l = 2$ and long $l = 5$ lead time values. Additionally, we set the holding costs at $h = 1$. Therefore, the chosen penalty parameters cover three different cases - when the holding costs result in 50% ($p = 1$), 10% ($p = 9$) and 3% ($p = 27$) of the total inventory system costs. The demand values are sampled from the Poisson distribution $D(t) \sim \text{Poiss}(3)$.

The lead time and penalty parameters for the backlog problem do not affect the performance of the optimal policy. The optimal policy parameters can be calculated for any combination of these settings. In the RL context, however, the delay (lead time) between action and reward and the variance of the rewards (penalty cost) influences the convergence properties of the RL algorithms. More specifically, highly volatile and delayed rewards are associated with poor algorithm performance.

On the other hand, the complexity of the problem rises with the increase in the lead time parameter for the lost sales. The reason for this is that the optimal policy depends on the full vector of outstanding orders. Additionally, higher penalty costs increase optimal inventory levels and decrease the probability of a stock-out. As a consequence, the base-stock policy is asymptotically optimal when the penalty parameter is large [22]. The combination of parameters (l and p) specified above augments the test cases presented in the previous study on lost sales problem and results in a wide range of scenarios [20].

Altogether, eighteen different benchmarks are formed, nine for the backlog and nine for the lost sales problems, respectively. These settings are also suitable for evaluating the benefits of reward variance reduction with Generalized Advantage Estimation (GAE). Thus, two PPO variations for all scenarios are tested: first, where only an actor-network is used, and second, where GAE is employed for calculating the advantages.

4.1.1 Backlog

The performance comparison between optimal base-stock policy and PPO policy is shown in Figure 4.2. The cost of the base-stock policy is normalized at 0 in order to be able to compare the results across the scenarios. The black error bars represent a 95% confidence interval, while the positive values of both the green and the red bars indicate the relative increase in the total costs, expressed in percentages.

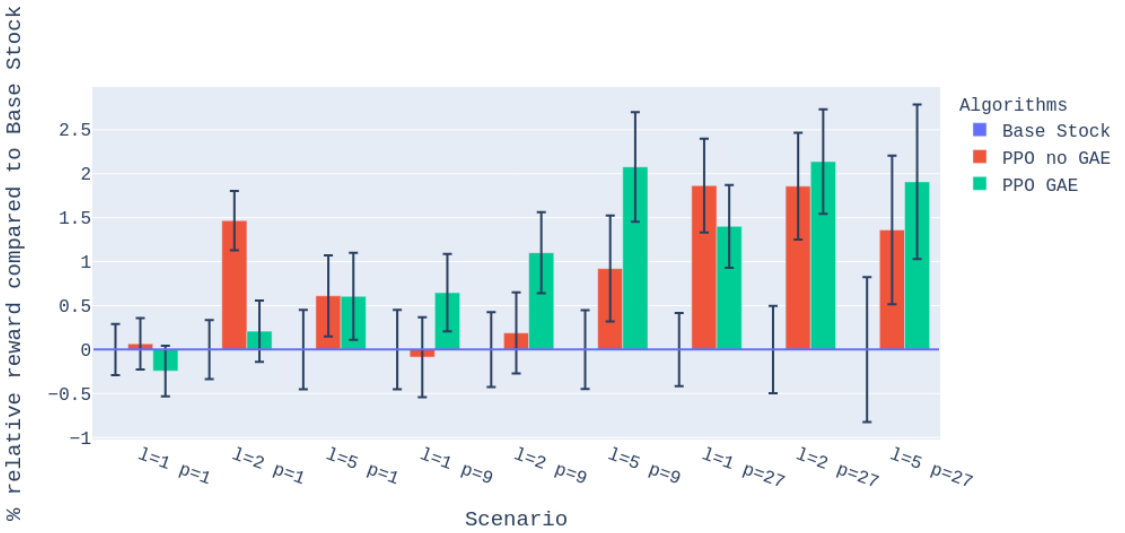


Figure 4.2: Mean costs difference in percentages on a single node system with backlog assumption, normalized with respect to the costs of the optimal base-stock policy (the lower is better). Obtained with $N = 1000$ iterations of $T = 100$ time-steps. Black error bar indicated the 95 % confidence interval

The figure shows that when the penalty parameter is low ($p = 1$), PPO (both GAE and no GAE) can find a nearly optimal policy. Furthermore, the graph shows a clear trend when the penalty costs are set to 9 ($p = 9$), and the lead times gradually increase ($l = [1, 2, 5]$). When $p = 9$, the subsequent growth in the lead time l causes PPO costs to rise and thus perform worse than the base-stock. We can see that the PPO with GAE experiences greater costs increase. The high reward variance can explain both cases: the stock-outs are frequent due to the average penalty. However, as the lead time grows, the uncertainty of the system increases since agent needs to plan many steps in advance. However, once the penalty costs are at its highest $p = 27$, there is no apparent change that comes from the change in the lead times, and the PPO's performance does not decrease. It is more beneficial to maintain high inventory levels when the penalties are large. Thus, the stock-outs are rare, and the PPO agent performance is not sensitive to the lead

time. In this case, the GAE and no GAE perform similarly, which is within the standard error.

Nevertheless, the results indicate that PPO can discover optimal or close to optimal policies (less than 2% gap) in single-echelon inventory systems under different lead time and penalty parameters. The results are in line with the expectation that with higher penalty parameters, the RL methods perform worse due to high reward variability directly influenced by the parameter p. Tuning the λ parameter for GAE did not lead to better results. Overall, the performance of PPO with GAE decreases slightly on high lead time and penalty parameters.

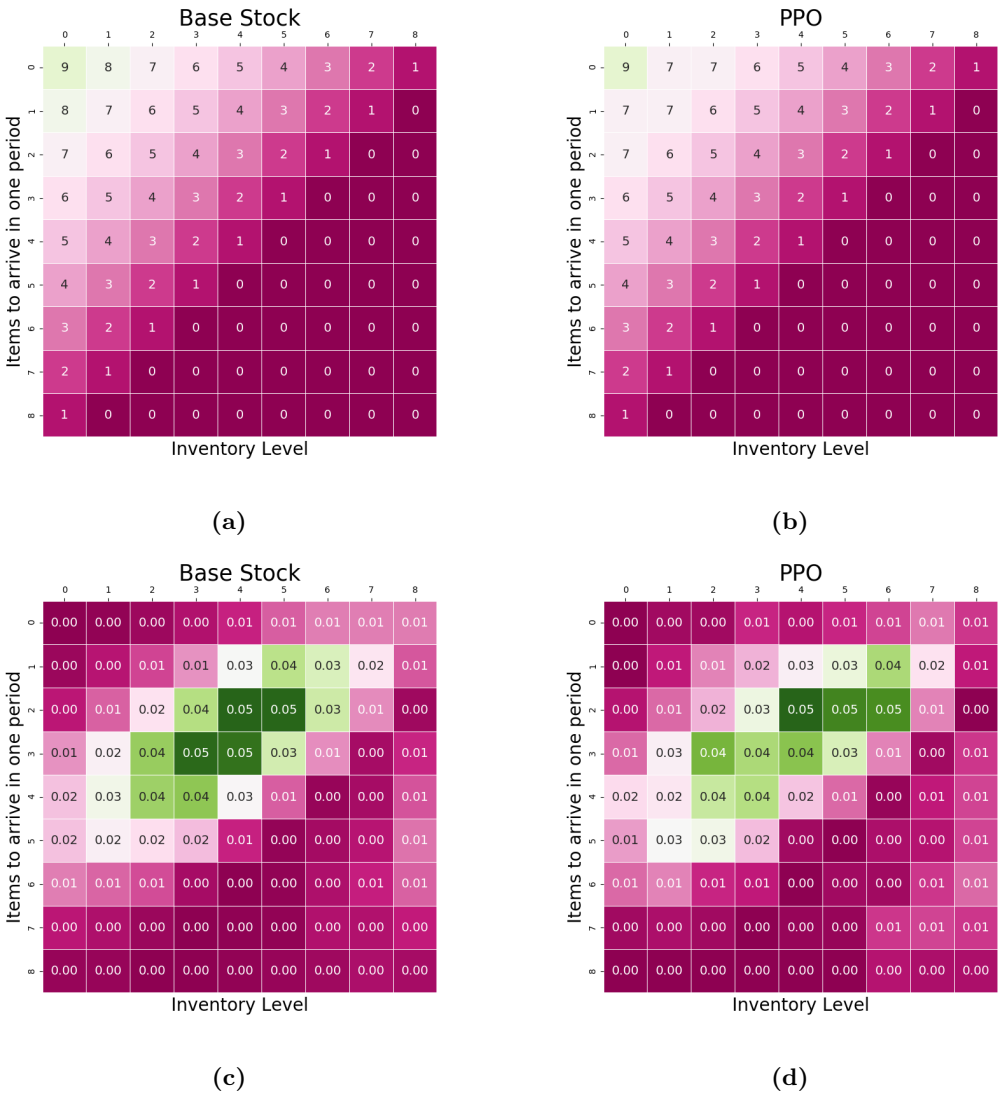


Figure 4.3: Visualization of the Base Stock and PPO policies and steady state inventory probabilities for single echelon backlog l=1, p=9, $D(t) \sim$ Poiss(3). Top: replenishment policies. Bottom: probability of state values in the steady state

Figure 4.3 provides visualizations of the base-stock and PPO replenishment policies. The

inventory level IL is positioned on the x-axis, while the y-axis indicates the number of outstanding orders $|Q|$. Sub-figures 4.3a and 4.3b contain the corresponding order quantities, while sub-figures 4.3c and 4.3d show the steady state probabilities. The underlying inventory model operates under the backlog assumption with the following parameters: $l = 1$, $p = 9$, $D(t) \sim$ Poiss(3). The base-stock levels are 9, and the PPO was trained without GAE.

The heat-map shows that the results of PPO are nearly identical to the base-stock policy. This means that the PPO agent is able to discover the optimal policy from the simulation data, which proves its potential for solving inventory optimization problems. The optimal policy and the policy obtained by PPO disagree on two states only: first, when the number of outstanding orders equals 0, and the inventory level equals 1; and second, when the outstanding orders equal 1 and the inventory level is 0. However, these states do not occur in the steady-state (bottom sub-figures), because both policies do not let the inventory position decrease below 1. It should be noted that when the PPO agent is given a state that lies outside of the steady-state distribution, it may output sub-optimal actions. This is an inherent attribute of the policies derived by the RL methods - not all possible states are visited or evaluated.

PPO is capable of discovering close to optimal policies on scenarios with short lead times and penalty costs. Additionally, a high penalty and long lead times result in an approximate 2% cost increase with respect to the optimal solution. Training PPO with GAE does not appear to be beneficial for the single echelon models with backlog - in some cases, it results in unstable training and higher costs.

4.1.2 Lost Sales

The results for the single-echelon model under the lost sales assumption are presented in Figure 4.4. The PPO agents can discover significantly better policies than the base-stock policy across almost all scenarios. The only exception is when $l = 1$ and $p = 27$. In this case, the PPO matches the base-stock policy, close to the optimal approach due to the short lead time and the high penalty cost. A3C's performance on the lost sales problem has been reported to be worse than that of the capped base-stock [20]. In contrast, our results indicate that PPO is able to outperform the capped base-stock on some settings ($l = 1, p = 1$ and $l = 5, p = 27$) and reach nearly equal performance in the remaining settings. The cost of the system under the capped base-stock policy is known to be close to optimal (less than 2% optimality gap) [20]. This allows us to conclude that PPO achieves close to optimal performance on the lost sales inventory optimization problem.

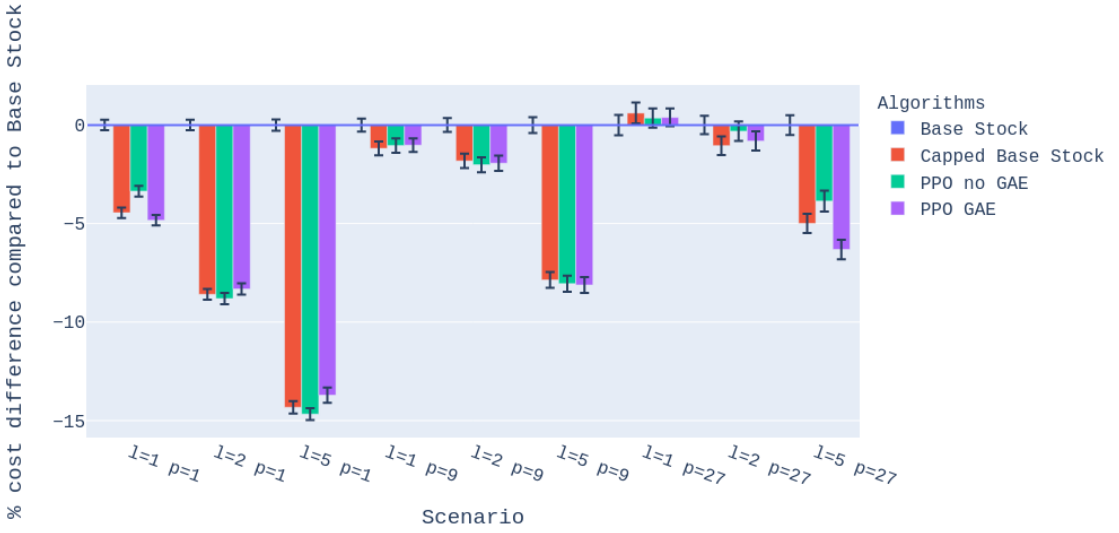


Figure 4.4: Comparison of the mean cost for different policies on single node system with lost sales assumption (the lower is better). Obtained under with $N = 1000$ iterations of $T = 100$ time-steps. Black error bars indicate the 95 % confidence interval

Moreover, PPO is stable to changes in lead time and penalty parameters. In contrast to the results from its backlogging counterpart, PPO with GAE was able to outperform the capped base-stock with a small margin (around 1%), when $p = 27$ and the reward variance are the highest. This result suggests that the benefits of smoothing the advantage estimates with GAE become more pronounced as the uncertainty of the environment increases. Therefore we expect GAE to be useful for the multi-node inventory systems.

The visualization of the capped base-stock and policy trained with PPO and corresponding probabilities of the steady-state values are presented in Figure 4.5. From the heat-maps, we can see that the policy trained with PPO is similar to the capped base-stock policy, with a small variation on the states with low occurrence probability in the steady-state. The differences between the policies (inventory level between 6 and 8, number of outstanding orders between 0 and 3) suggest that PPO tends to order a single additional item when the inventory levels are high, and the number of outstanding orders is low. However, these states are rare, therefore the total costs of the both policies are within the confidence interval, shown in Figure 4.4 where $l = 1$, $p = 9$. Nevertheless, this observation suggests greater flexibility of neural network-based policy. In the instances with the highest lead time and penalty costs, the PPO agent outputs higher order quantities than otherwise allowed by the capped base-stock, which could be a source of improvement.

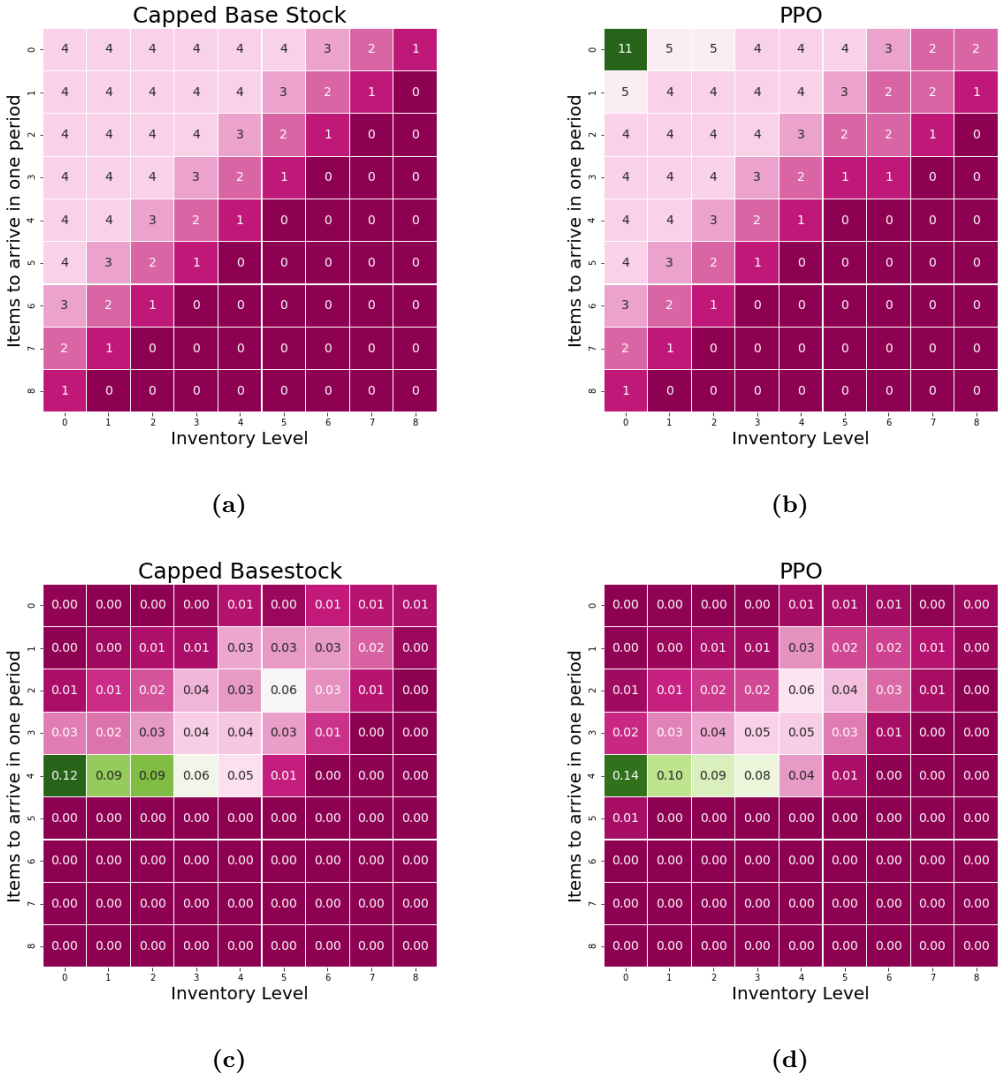
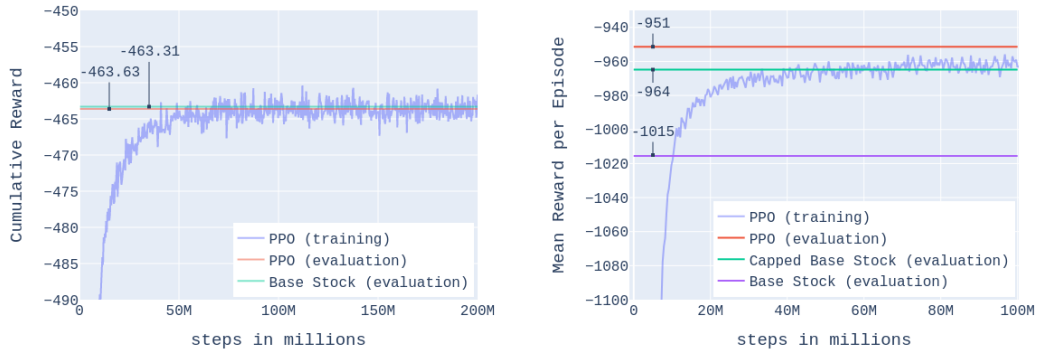


Figure 4.5: Visualization of the Capped Base Stock and PPO (with GAE) policies and steady state inventory probabilities for single echelon lost sales l=1, p=9, $D(t) \sim$ Poiss(3). Top: replenishment policies. Bottom: probability of state values occurring in the steady state

4.1.3 Computational Complexity

Figure 4.6 shows the training curve of the PPO algorithm in contrast with the heuristic policies, which helps us evaluate the computational efficiency of the PPO method. PPO was able to achieve a reasonable performance within 50 million steps, which corresponds to 500,000 training episodes. The number of steps is high due to the large batch size of 32,768. Each 200 million steps correspond to 16 hours of wall-clock time on 12 logical CPU (clock speed 3.7 GHz) threads. Training on a smaller batch size slowed down the training without any convergence improvements. Large training batch sizes were also used to decrease the number of training iterations and improve



(a) Single Echelon backlog ($l = 1$, $p = 9$) **(b)** Single Echelon lost sales ($l = 5$, $p = 27$)

Figure 4.6: Selected training runs for Single Echelon

training stability.

We can conclude that PPO is a computationally expensive method that requires a high number of training samples. However, this conclusion refers to the initial computational costs. We expect diminishing requirements of the number of samples as the problem becomes more difficult (increasing number of locations).

4.2 Multi Echelon

This section covers the experiments performed on the One-Warehouse Multi-Retailer problem (Figure 4.7) introduced in Chapter 3. Here we investigate how different demand processes and different lead times at the retailers affect the performance of PPO in comparison to the heuristic policies.

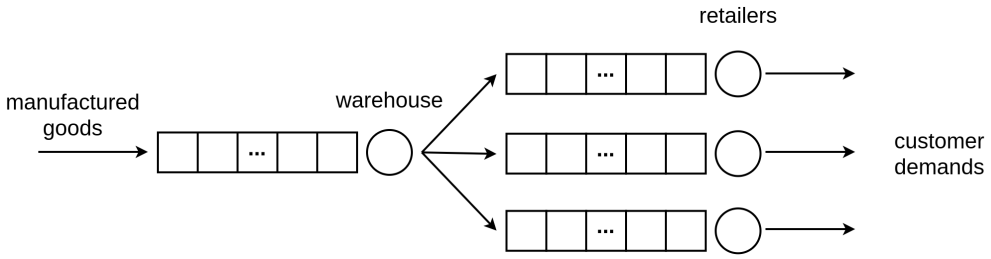


Figure 4.7: Illustration of buffers in one-warehouse multi-retailer systems with lead times

One Warehouse Multiple Retailers (OWMR) pose a problem where a large amount of possible actions is available. Previous studies [61, 20] have aimed to reduce the action space by exploiting

the system's symmetry. Neural networks addressed the problem of large state spaces. However, large action spaces are usually addressed by restricting the number of actions for a certain number. In contrast, our work presents an approach that tackles large action spaces without the need for a penalty for infeasible actions. Our method is capable of controlling all stages in the one-warehouse multiple-retailer problem individually.

We evaluate ten scenarios, separated into two categories, namely symmetric and asymmetric, with three main attributes:

- demand processes D_1, D_2, D_3
- lead times l_0, l_1, l_2, l_3
- customer behavior

The summary of the scenarios is presented in Table 4.1 below.

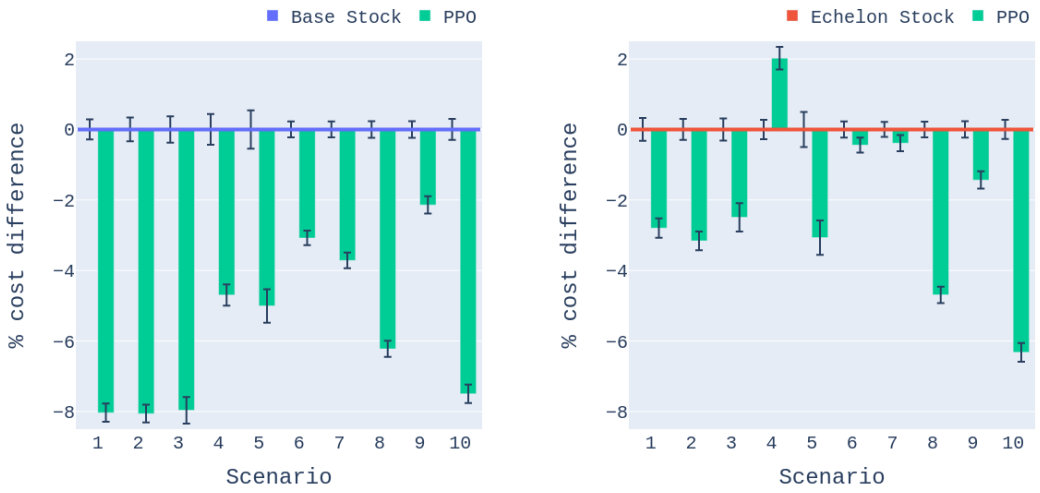
Scenario	Customer Behaviour	l_0	l_1	l_2	l_3	D_1	D_2	D_3
1	backlog	2	1	1	1	Poiss(3)	Poiss(3)	Poiss(3)
2	backlog	2	1	1	1	$\mathcal{U}(0,6)$	$\mathcal{N}(3,1)$	Poiss(3)
3	backlog	2	1	1	1	$\mathcal{N}(1,5)$	$\mathcal{N}(5,1)$	Poiss(0.5)
4	backlog	2	1	2	3	Poiss(3)	Poiss(3)	Poiss(3)
5	backlog	5	3	3	3	$\mathcal{N}(1,5)$	$\mathcal{N}(5,1)$	Poiss(0.5)
6	lost sales	1	1	1	1	Poiss(3)	Poiss(3)	Poiss(3)
7	lost sales	2	1	1	1	Poiss(3)	Poiss(3)	Poiss(3)
8	lost sales	5	1	1	1	Poiss(3)	Poiss(3)	Poiss(3)
9	lost sales	2	1	2	3	Poiss(3)	Poiss(3)	Poiss(3)
10	lost sales	5	3	3	3	$\mathcal{N}(1,5)$	$\mathcal{N}(5,\ 1)$	Poiss(0.5)

Table 4.1: Different scenarios tested for one-warehouse multi-retailer

We differentiate between symmetric (*1, 6, 7, 8*) and asymmetric (*2, 3, 4, 5, 9, 10*) scenarios. More specifically, scenarios *1, 6, 7, 8* represent symmetric systems, where all the retailers share the same attributes. Scenarios *2, 3, 10*, on the other hand, are asymmetric with different demand processes at the retailers; while the asymmetry in *4 and 9* comes from the different transportation lead times of the retailers. Finally, we distinguish between scenarios *1-5* and *6-10* based on the backlog and lost sales attribute. Scenarios *1-5* assume full backlogging at the retailers, while for *6-10* the unmet demand is lost. Altogether, these settings allow for the performance evaluation of the PPO algorithm under different sources of asymmetry.

The results across all scenarios are presented in Figure 4.8. The bar plot follows the same scheme as in the previous section: the heuristics costs are set to 0, while the costs of the PPO agents are expressed as a difference in percentages. First, we analyze the symmetric systems under

the backlog and lost sales assumptions. Further, the selected asymmetric scenarios are evaluated based on the source of the asymmetry.



(a) Costs of PPO normalized with respect to the costs of base-stock heuristic policy

(b) Costs of PPO normalized with respect to the costs of echelon-stock heuristic policy

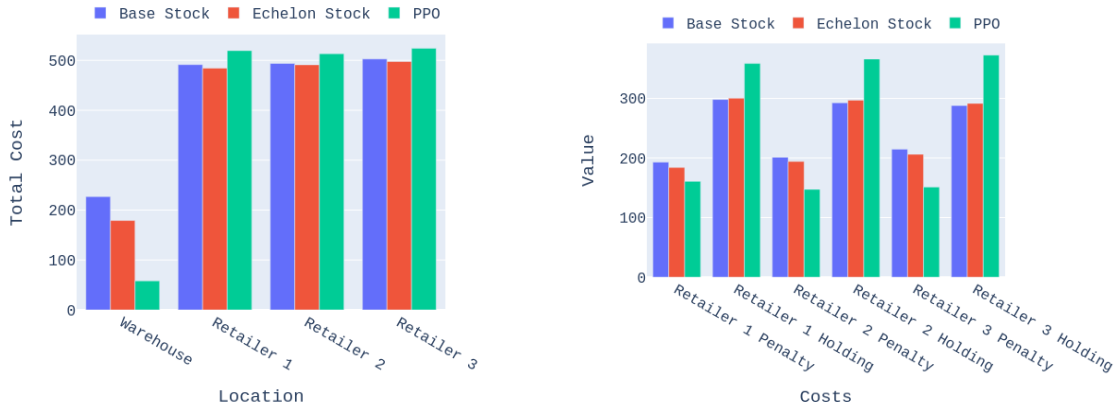
Figure 4.8: The average costs per episode $T = 100$ calculated across $N = 1000$ iterations. The green bars represent the decrease in total costs of the inventory system, when controlled by a policy optimized by the PPO algorithm (the lower is better). Black error bars indicate 95% confidence interval.

4.2.1 Symmetric Systems

Scenario 1. We begin with evaluating the PPO performance on the symmetric OWMR models with identical retailers. This setting is characterized by short retailers $l_{1,2,3} = 1$ and warehouse $l_0 = 2$ lead times, and a low coefficient of variation of the demand at the retailers $CV = 0.57$. We can see from Figure 4.8a that the PPO agent manages to outperform the base-stock policy by 8% and the echelon stock policy by more than 2% (figure 4.8b). This shows that the DRL approach can develop policies that outperform the heuristics in symmetric OWMR models with short lead times.

The reason for these results can be seen in Figure 4.9a. While the total costs of the retailers increase slightly in comparison with the heuristics, the warehouse holding costs are considerably reduced. Figure 4.9b provides a detailed breakdown of all associated costs per location. The holding costs of the retailers are higher, yet the penalty costs have decreased. This indicates that the PPO agent tends to transfer more items to the retailers in order to reduce the holding costs at

the warehouse. However, the costs for the retailers are raised by about 1.5% each, while the costs of the warehouse are reduced by 11.5% in comparison with the total cost of the system controlled by the base stock heuristics.



(a) Aggregate costs per location

(b) Costs per location.

Figure 4.9: Costs for Scenario 1.

To investigate this issue further, Table 4.2 shows the comparison between the mean on-hand inventory before and after the allocation decision has taken place.

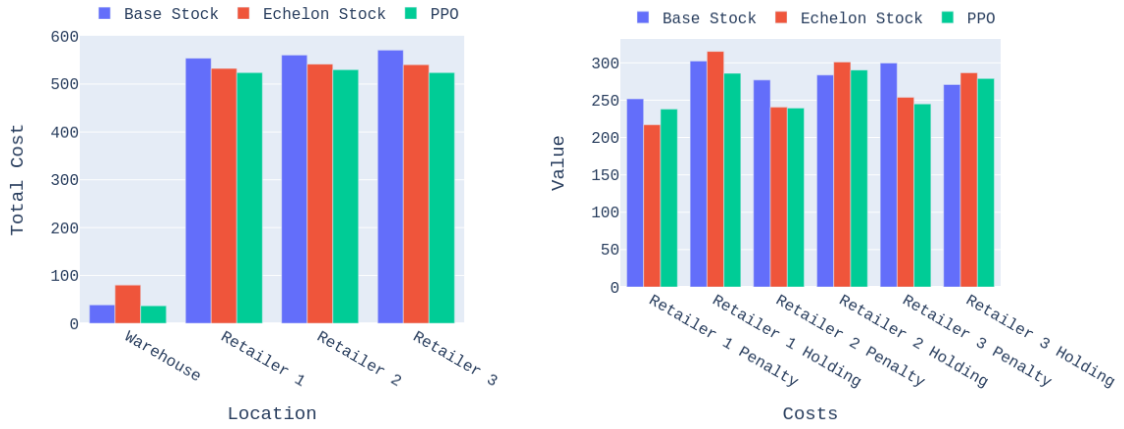
	Base Stock	Echelon Stock	PPO
On-hand Before Allocation	13.73	12.64	10.17
On-hand After Allocation	4.73	3.69	1.22
Shortage	0.38	0.40	0.18

Table 4.2: Mean on-hand inventory before allocation, on-hand inventory after allocation and shortage on the warehouse under the heuristics and the PPO policies in Scenario 1. The episode length is increased to $T = 10000$ to provide more accurate estimates.

The PPO agent manages to maintain a smaller on-hand inventory, while lowering the shortage at the warehouse. The RL agent could achieve this by tracking the number of on-hand inventory at the warehouse and then efficiently allocating those items across the retailers. The RL agent shows that by jointly optimizing all of the locations it can achieve significant cost savings with respect to the best reorder point policies.

Scenarios *6,7,8* present symmetric systems under the lost sales assumption with increasing warehouse lead times ($l_0 = [1, 2, 5]$ respectively). According to Figure 4.8 presented earlier, the performance of PPO does not differ from the echelon stock in scenarios *6, 7*. The cost difference in these cases is less than 0.5%. For this reason, we do not analyze these scenarios in detail.

As the lead time increases to $l_0 = 5$, we notice an improvement in Scenario *8* in comparison to the echelon stock heuristics. Figure 4.10a presents the visualization of the aggregate costs per each location. Unlike in the backlogging counterpart (Scenario *1*), the cost improvements are driven by the retailers. The warehouse cost remains at the same level as the base-stock heuristic, meanwhile, the costs of the retailers decrease by approximately 6%. Furthermore, although the echelon stock achieves better results in lowering the costs at the retailers with a 3% improvement, it doubles the holding costs at the warehouse compared to the base stock. We can conclude that PPO can maintain significantly lower costs in the OWMR system under lost sales compared to the heuristic policies, when the production lead time is high (6% cost decrease compared to base stock and 4% compared to echelon stock).



(a) Aggregate costs per location

(b) Costs per location.

Figure 4.10: Costs for Scenario 8.

4.2.2 Asymmetric Systems

Scenarios (2, 3) present the cases where the demand distribution of the retailers changes. In Scenario *2*, the mean of the retailers' demand is kept at approximately the same level ($\mu_j = 3$, $j = 1, 2, 3$) as in *1*, and only the type of the distribution is modified. The costs analysis is analogous to Scenario *1*, thus, a further explanation is deemed redundant. Cost plots are provided in Appendix B (Figures B.2 and B.3).

In Scenario *3*, the parameters are adjusted as to introduce an imbalance: the mean of the retailers demand is selected so that the corresponding coefficient of variation ($CV = \frac{\sigma}{\mu}$) is high on the first and third retailers ($CV_1 = 1.37$, $CV_3 = 1.41$) and low on the second one ($CV_2 = 0.23$).

The first retailer demand is sampled from normal distribution $\mathcal{N}(1, 5)$, with high variance and low mean. The second retailer demand is sampled from a more stable normal distribution with higher mean and lower variance $\mathcal{N}(5, 1)$. Finally, the third retailer has the lowest demand, according to Poiss(0.5). This setting is designed to test the resilience of our method to the varying demand of the retailers. Despite the increase in the coefficient of variation, the PPO performance remains the same compared to the heuristics in the previous two scenarios.

Scenarios *4, 9* represents asymmetric systems where the asymmetry is introduced by the difference in the retailers' lead time. Scenario 4 presents a setting under the backlog assumption and Scenario *9* under lost sales.

The PPO agent in scenario *4* reduces costs by approximately 5% with respect to the base stock. However, in comparison to the echelon stock, PPO produces worse results, as the echelon-stock results in additional 2% cost improvements. The inferior performance of PPO is related to the training process of the agent. All replenishment decisions, in this case, are produced at once with the sequential allocation rule. Thus, if action is infeasible (i.e., the sum of the retailers' orders is more than the on-hand inventory at the warehouse), the last retailer in the random sequence receives the lowest amount of items. This approach increases the costs at that retailer and lowers the rewards for the PPO agent, which, in turn, decreases the probability of the infeasible actions. Furthermore, the introduction of skewing the retailers' lead times causes a part of the reward to be delayed, which could explain the lower performance of PPO.

This issue persists in scenario *9*. Despite this caveat of our method, PPO manages to discover policies that achieve a 2% cost reduction with respect to the base stock and a 1.3% reduction with respect to the echelon stock. The costs are illustrated in Figure B.9 in Appendix B.

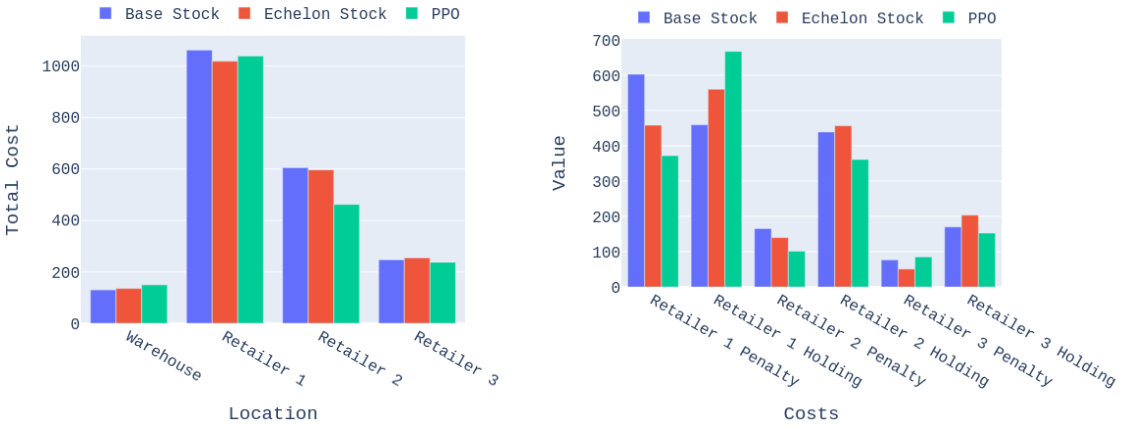
Scenario *5, 10* combine long lead times at the retailers $l_{1,2,3} = 3$ and the warehouse $l_0 = 5$, with asymmetrical distributions of the retailers and the same coefficient of variations as in scenario 3 ($CV_1 = 1.37$, $CV_2 = 0.23$, $CV_3 = 1.41$). In both scenarios *5* and *10*, PPO managed to reduce the costs of the second retailer, with the lowest demand variability and the highest demand mean. The costs for scenarios *5* and *10* are summarized in Figure 4.11 and 4.12 respectively.

In both cases, the PPO agent has managed to improve the holding costs at the second retailer. In scenario *10*, in particular, the PPO agent has managed to decrease the penalty costs where the lost sales assumption is employed. This could be explained by the steadier orders produced by the PPO agent.

The histograms on Figure 4.13 show the distribution of the agents' orders. The horizontal axis

(a) Aggregate costs per location **(b)** Costs per location.

Figure 4.11: Costs for Scenario 5.

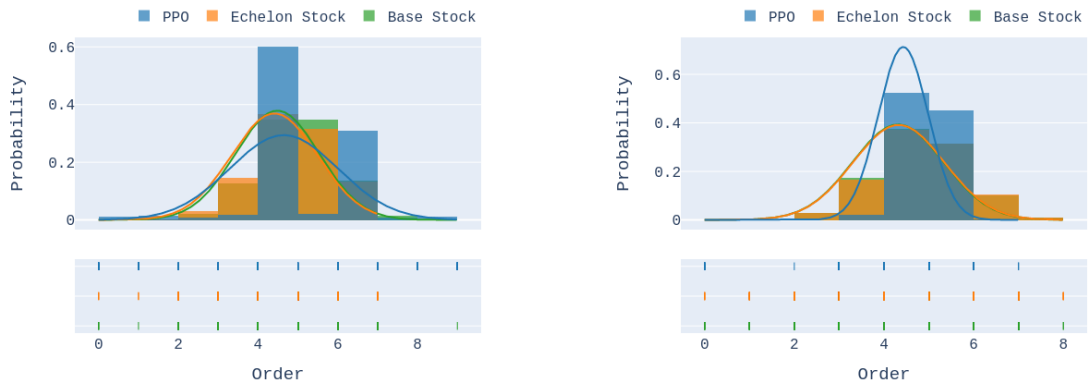


(a) Aggregate costs per location **(b)** Costs per location.

Figure 4.12: Costs for Scenario 10.

indicates the order size, while the vertical axis shows the probability of the agent placing the order. The histograms of the base stock and the echelon stock are nearly identical as it can be seen in the histogram - the green and the yellow lines are overlapping each other. It can be noticed that the PPO agent places orders in a narrower range compared to the heuristics. The most common actions for the backlog problem are 4 and 6 (scenario 5), and for the lost sales problem, these are 4 and 5 (scenario 10). The PPO agent is able to stabilize the orders at the biggest retailer to ensure a total cost reduction across the system.

In the case of the first retailer with demand $D_1(t) \sim \mathcal{N}(1, 5)$, the PPO agent ensures higher



(a) Scenario 5. **(b)** Scenario 10.

Figure 4.13: Histogram of action values for the Retailer 2

stock to reduce the potential penalty resulting in higher overall costs. Nevertheless, the costs savings of the PPO agent are 6 % compared to echelon-stock and almost 8 % with respect to the base-stock.

4.2.3 Summary

Overall, the policies trained with the PPO algorithm can outperform the well-established heuristics on a range of multi-echelon inventory optimization problems. These include systems with symmetric retailers and systems with asymmetric retailers related to associated demand process (distribution, mean, and variance). Additionally, the RL agent can discover well-performing policies for the multi-echelon systems with lost sales and long retailers' lead time. More specifically, the results from scenarios *1,2,3,5,8,10* indicate that the PPO algorithm can outperform the echelon-stock heuristic policy by a significant margin: 2% cost savings on symmetric problems and up to 6% on asymmetric problems with long lead times.

The neural network with several action heads has proven to be suitable for large action spaces in the context of the OWMR systems. Despite the fact that problems with asymmetric retailers' lead times presented in scenarios *4* and *9* are a challenging task for our solution method, we have successfully shown that PPO is able to perform better than the widely accepted heuristic on a series of asymmetric OWMR problems.

4.2.4 Influence of GAE on training process

During the experiments on OWMR problems, several variations of the network architecture and hyper-parameters were tested. The best results were obtained utilizing GAE during training. The PPO agent was trained with GAE on all of the OWMR problems presented in this section.

We evaluate the effect that GAE has on the performance. Figure 4.14 presents the PPO algorithm trained for scenario *4* with different variance reduction techniques. Here we can see that the performance of PPO with GAE improves in comparison to PPO with advantage critic and PPO with no critic at all. This shows that GAE is an important component when training PPO on complex inventory problems: single echelon lost sales with long lead times and high penalty costs and OWMR systems.

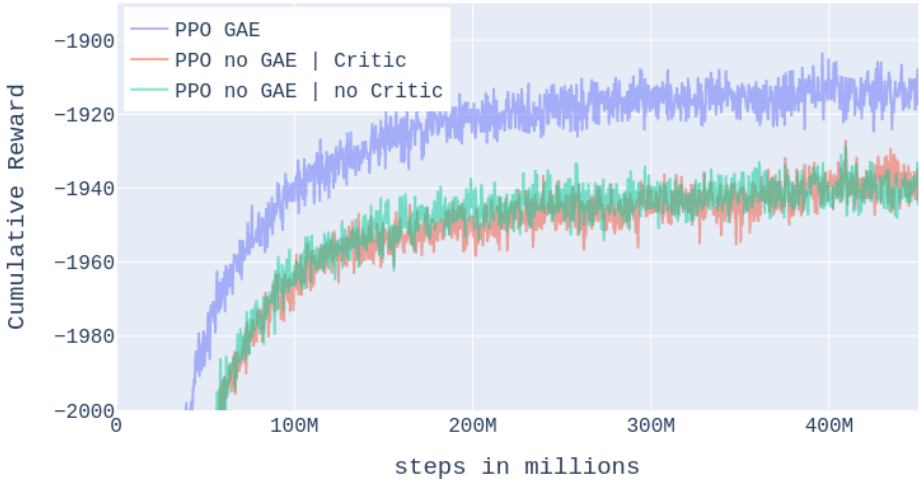


Figure 4.14: Comparison of PPO training (OWMR Setting *4*.) with different settings: 1) PPO GAE - Generalized Advantage Estimation is used with $\lambda = 0.98$. 2) PPO no GAE - regular advantage estimation is used for the critic. 3) PPO no Critic - only actor network is used

4.2.5 Computational Complexity

Figure 4.15 presents the training curve for the PPO. It can be noticed that the RL agent starts to outperform the base stock heuristic at around 50 million and the echelon stock at around 180 million training steps. As in the case of a single echelon, the training was performed on 12 CPU cores, where 200 million steps correspond to 16 hours of wall clock time. Compared to the single echelon systems, the number of state and action dimensions has increased 3-fold. The total

number of steps required to reach the performance of the base-stock heuristic did not change. The echelon stock heuristic was proven to be a more challenging baseline that required PPO 4 times more training time. Nevertheless, the flexibility of DRL methods allows tackling the more complex problem: capacity constraints, emergency shipments, non-linear costs, additional transportation routes can be easily integrated into the simulation environment.

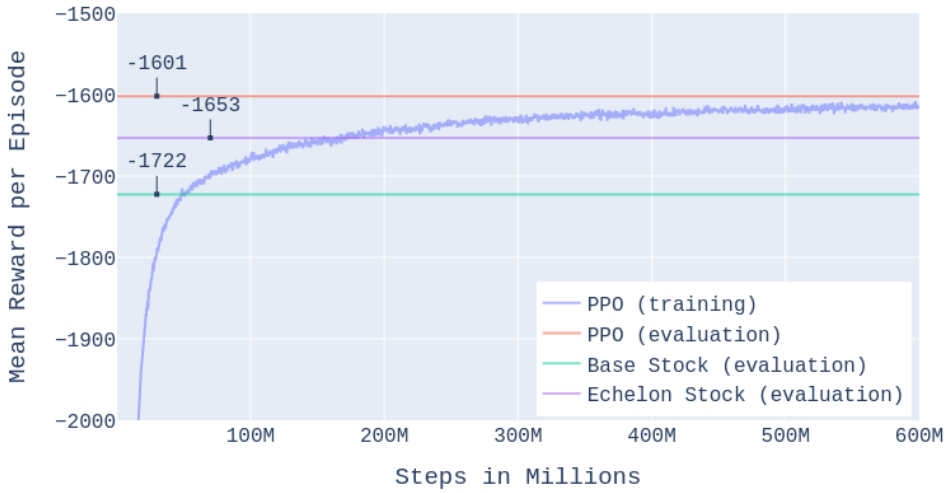


Figure 4.15: Training/evaluation of PPO versus the evaluation of base-stock and echelon-stock heuristics, scenario *1*.

Chapter 5

Conclusions and Future Research

5.1 Conclusions

We have presented an application of a novel method for stochastic control in inventory management in this work. Proximal Policy Optimization (PPO), a popular Deep Reinforcement Learning (DRL) algorithm, was used to derive replenishment policies for single echelon and multi-echelon inventory systems. We have evaluated the effects of different problem parameters on the performance of neural network-based policies compared to the widely accepted base-stock and echelon-stock heuristics. The PPO method has proven to be a suitable candidate for stochastic inventory optimization. Further, we address the research questions presented in chapter 1.

> *RQ T.1: How can Deep Reinforcement Learning methods be applied to optimizing the replenishment decisions for single node inventory models?*

We have applied a Multi-layer Perceptron trained with PPO and achieved close to optimal costs in a single-node inventory system under the backlog assumption. For the lost sales problem, our method has proven to be competitive with the strong heuristic policies and outperformed previous DRL benchmarks [20]. Additionally, in our study, we have identified that Generalized Advantage Estimation (GAE) affects training positively in the lost sales problems with long lead times and high penalty values.

> *RQ T.2: How can Deep Reinforcement Learning methods be applied to simultaneous control of replenishment decisions on multiple locations?*

Our main contribution lies within the more complex inventory optimization problem, one-warehouse multi-retailer (OWMR). In this case, we have shown that PPO is a suitable candidate for training a neural network to control inventory levels of multiple locations simultaneously. We used a single Multi-layer Perceptron and processed its output by several softmax activation functions for specifying the exact replenishment decisions for all locations in OWMR. A specific training procedure was designed to improve the policy search without manually adjusting the penalties for infeasible actions. Moreover, the inventory models used in this study do not include capacity constraints, emergency shipments, or partial lost sales to increase the competitiveness of the baseline heuristic policies. Therefore, the s-type heuristics are widely accepted as a strong baseline in the problems we consider.

RQ T.3: How well can Deep Reinforcement Learning perform on asymmetric one-warehouse multi-retailer stochastic inventory optimization problems compared to the well-established s-type heuristic policies?

We evaluated the proposed method across ten different scenarios. The PPO agent outperformed the base-stock heuristic policy in every scenario. For the symmetric systems, the cost improvements were up to 8%. The cost savings with respect to the echelon-stock heuristic were up to 6% for the most complex system with long lead times and lost sales. The echelon stock heuristic outperformed our approach only in one case, scenario four, where the retailer's lead times differ. Nevertheless, the method employed in this study showed consistent improvements over the widely accepted heuristic policies.

RQ: How can Deep Reinforcement Learning contribute to reducing system-wide operational costs in asymmetric one-warehouse multi-retailer stochastic inventory optimization problems?

This study evaluated the suitability of PPO for training neural networks to control stochastic inventory optimization problems. The problems' complexity was gradually increased from single-node systems under backlog/lost sales assumptions to the multi-echelon divergent systems. We have tested multiple problem instances under which the DRL agent could outperform the heuristics on most of the problem settings. The heuristic policies formed a solid baseline since we have adjusted the problem formulation to ensure a fair comparison. The echelon stock heuristic has proven to be a better choice for only one OWMR problem setting system with a backlog and asymmetric lead times.

We have addressed the limitations of previous works [61, 20] in controlling the asymmetric OWMR models, where retailers have different demand distribution. Our method was able to scale on the problem instances with long lead times of the warehouse and the retailers. We achieved these results by employing a Deep Reinforcement Learning method for simultaneous control of the replenishment decisions of all the locations. The general nature of the approach has allowed us to use our method for all of the scenarios without hyper-parameter fine-tuning, reward scaling or state prepossessing. All of the above allows us to conclude that PPO is a suitable algorithm for stochastic inventory optimization.

5.2 Limitations and Future Research

The most notable limitation of our method is its inability to scale on asymmetric OWMR models with different retailers' lead times (shown in scenario four in Chapter 4). Future research should consider addressing this issue by using a different approach to handling infeasible actions. For example, the allocation decision can be formed by identifying replenishment orders one by one for each retailer. This will increase the computational requirements since we need to query the policy network several times at every time-step. At the same time, action masking can be used to disallow the choice of infeasible actions.

Another possible direction for future research is to convey information about the system state in a graph form. Graph Convolution Networks [32], in general, and Graph Convolution Reinforcement Learning [25], in particular, are suitable approaches to include the problem structure into the system state. The work of [25] considers multi-agent environments that are represented as a graph, where nodes are agents cooperating/competing with each other. This might improve the agents' awareness about other actors in the system, improve cooperation, and scale the method on larger problem instances. Moreover, Graph Convolutions would allow for processing any type of Supply Chain Network: general, divergent, serial, or convergent.

Another approach is to form a hybrid method between RL and the heuristic policies. Reinforcement Learning agent dynamically specifies well-performing s-type heuristics at every decision period. This could be done in two different ways. First, the output of the RL agent can be the parameters of the heuristic policy. Similar to the approach presented in [61, 20], but the heuristics can be specified individually for each location. Second, a set of well-performing heuristics can be formed (i.e., multiple constant or base-stock heuristics), and the task of the RL agent is to choose the most suitable heuristic based on the state information. [46] presents an example of such an

approach, which could be adapted to the current problem.

Another direction is to explore paired optimization. In this example, the Reinforcement Learning algorithm and the appropriate baseline are considered. If the action of the RL agent deviates greatly from the baseline an additional term is added to the loss similar to [39].

Bibliography

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019. 2

[2] Kenneth Joseph Arrow. Studies in the mathematical theory of inventory and production. 8

[3] Sven Axsäter. *Inventory control*, volume 225. Springer, 2015. 5, 8, 35

[4] Sven Axsäter and Kaj Rosling. Installation vs. echelon stock policies for multilevel inventory control. *Management Science*, 39(10):1274–1280, 1993. 35

[5] Bharathan Balaji, Jordan Bell-Masterson, Enes Bilgin, Andreas Damianou, Pablo Moreno Garcia, Arpit Jain, Runfei Luo, Alvaro Maggiar, Balakrishnan Narayanaswamy, and Chun Ye. Orl: Reinforcement learning benchmarks for online stochastic optimization problems. *arXiv preprint arXiv:1911.10641*, 2019. 17

[6] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. 2

[7] Richard Bellman. An introduction to the theory of dynamic programming. Technical report, RAND CORP SANTA MONICA CA, 1953. 21

[8] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 2

[9] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995. 10

[10] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996. 2

[11] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 36

[12] S Kamal Chaharsooghi, Jafar Heydari, and S Hessameddin Zegordi. A reinforcement learning model for supply chain ordering management: An application to the beer game. *Decision Support Systems*, 45(4):949–959, 2008. 12

[13] Andrew J Clark and Herbert Scarf. Optimal policies for a multi-echelon inventory problem. *Management science*, 6(4):475–490, 1960. 6, 7

[14] Tapas K Das, Abhijit Gosavi, Sridhar Mahadevan, and Nicholas Marchalleck. Solving semi-markov decision problems using average reward reinforcement learning. *Management Science*, 45(4):560–574, 1999. 11

[15] Ton de Kok, Christopher Grob, Marco Laumanns, Stefan Minner, Jörg Rambau, and Konrad Schade. A typology and literature review on stochastic multi-echelon inventory models. *European Journal of Operational Research*, 269(3):955–983, 2018. 2, 3, 6

[16] Mustafa K Doğru, AG De Kok, and Geert-Jan van Houtum. A numerical study on the effect of the balance assumption in one-warehouse multi-retailer inventory systems. *Flexible services and manufacturing journal*, 21(3-4):114–147, 2009. 7, 8, 9

[17] Mehdi Firoozi. *Multi-echelon Inventory optimization under supply and demand uncertainty*. PhD thesis, 2018. 16

[18] J-P Gayon, Guillaume Massonnet, Christophe Rapine, and Gautier Stauffer. Constant approximation algorithms for the one warehouse multiple retailers problem with backlog or lost-sales. *European Journal of Operational Research*, 250(1):155–163, 2016. 2

[19] Ilaria Giannoccaro and Pierpaolo Pontrandolfo. Inventory management in supply chains: a reinforcement learning approach. *International Journal of Production Economics*, 78(2):153–161, 2002. 11

[20] Joren Gijsbrechts, Robert N Boute, Jan A Van Mieghem, and Dennis Zhang. Can deep reinforcement learning improve inventory management? performance on dual sourcing, lost

sales and multi-echelon problems. *Performance on Dual Sourcing, Lost Sales and Multi-Echelon Problems (July 29, 2019)*, 2019. ii, ii, 2, 3, 17, 18, 25, 27, 34, 36, 39, 42, 45, 55, 57

[21] Roger M Hill, Mehdi Seifbarghy, and David K Smith. A two-echelon inventory model with lost sales. *European Journal of Operational Research*, 181(2):753–766, 2007. 8

[22] Woonghee Tim Huh, Ganesh Janakiraman, John A Muckstadt, and Paat Rusmevichientong. Asymptotic optimality of order-up-to policies in lost sales inventory systems. *Management Science*, 55(3):404–420, 2009. 39

[23] Woonghee Tim Huh, Ganesh Janakiraman, and Mahesh Nagarajan. Capacitated multiechelon inventory systems: Policies and bounds. *Manufacturing & Service Operations Management*, 18(4):570–584, 2016. 2

[24] Ganesh Janakiraman and John A Muckstadt. A decomposition approach for a class of capacitated serial systems. *Operations Research*, 57(6):1384–1393, 2009. 3

[25] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. *arXiv preprint arXiv:1810.09202*, 2018. 57

[26] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002. 15

[27] Ahmet Kara and Ibrahim Dogan. Reinforcement learning approaches for specifying ordering policies of perishable inventory systems. *Expert Systems with Applications*, 91:150–158, 2018. 12

[28] SAMUEL Karlin and HERBERT Scarf. Inventory models and related stochastic processes. *Studies in the mathematical theory of inventory and production*, 1:319, 1958. 8

[29] Lukas Kemmer, Henrik von Kleist, Diego de Rochebouët, Nikolaos Tziortziotis, and Jesse Read. Reinforcement learning for supply chain optimization. In *European Workshop on Reinforcement Learning 14*, pages 1–9, 2018. 16

[30] Steven O Kimbrough, Dong-Jun Wu, and Fang Zhong. Computers play the beer game: can artificial agents manage supply chains? *Decision support systems*, 33(3):323–333, 2002. 12

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 14, 30

[32] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 57

[33] Anton J Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002. 8

[34] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062, 2018. 28

[35] Christos T Maravelias and Charles Sung. Integration of production planning and scheduling: Overview, challenges and opportunities. *Computers & Chemical Engineering*, 33(12):1919–1930, 2009. 8

[36] John T Mentzer, William DeWitt, James S Keebler, Soonhong Min, Nancy W Nix, Carlo D Smith, and Zach G Zacharia. Defining supply chain management. *Journal of Business logistics*, 22(2):1–25, 2001. 1

[37] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016. 2, 17

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 14

[39] Mohammadreza Nazari, Majid Jahani, Lawrence V Snyder, and Martin Takáč. Don't forget your teacher: A corrective reinforcement learning framework. *arXiv preprint arXiv:1905.13562*, 2019. 58

[40] Afshin Oroojlooyjadid, MohammadReza Nazari, Lawrence Snyder, and Martin Takáč. A deep q-network for the beer game: A deep reinforcement learning algorithm to solve inventory optimization problems. *arXiv preprint arXiv:1708.05924*, 2017. 16, 18

[41] Afshin Oroojlooyjadid, Lawrence Snyder, and Martin Takáč. Stock-out prediction in multi-echelon networks. *arXiv preprint arXiv:1709.06922*, 2017. 17

[42] Afshin Oroojlooyjadid, Lawrence V Snyder, and Martin Takáč. Applying deep learning to the newsvendor problem. *IISE Transactions*, 52(4):444–463, 2020. 17

[43] Jooyoung Park and Irwin W Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991. 17

[44] Sangwook Park. Routing and inventory allocation policies in multi-echelon distribution systems. 1998. 2

[45] Pierpaolo Pontrandolfo, Abhijit Gosavi, O Geoffrey Okogbaa, and Tapas K Das. Global supply chain management: a reinforcement learning approach. *International Journal of Production Research*, 40(6):1299–1317, 2002. 11

[46] Paolo Priore, Borja Ponte, Rafael Rosillo, and David de la Fuente. Applying machine learning to the dynamic selection of replenishment policies in fast-changing supply chain environments. *International Journal of Production Research*, 57(11):3663–3677, 2019. 57

[47] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. 14

[48] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015. 15, 27

[49] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. 31

[50] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2, 3, 15, 27, 32

[51] Joohyun Shin, Thomas A Badgwell, Kuang-Hung Liu, and Jay H Lee. Reinforcement learning–overview of recent progress and implications for process control. *Computers & Chemical Engineering*, 127:282–294, 2019. 9

[52] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017. 2

[53] Lawrence V Snyder and Zuo-Jun Max Shen. *Fundamentals of supply chain theory*. Wiley Online Library, 2011. 5

[54] Hiroshi Suetsugu, Yoshiaki Narusue, and Hiroyuki Morikawa. Joint replenishment policy in multi-product inventory system using branching deep q-network with reward allocation. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 115–121. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2019. 18

[55] Jiankun Sun and Jan A Van Mieghem. Robust dual sourcing inventory management: optimality of capped dual index policies and smoothing. *Manufacturing & Service Operations Management*, 21(4):912–931, 2019. 34

[56] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011. 9, 10, 11, 15, 18

[57] Richard S Sutton, Andrew G Barto, and Ronald J Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, 1992. 9

[58] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 18

[59] MC Van der Heijden, EB Diks, and AG De Kok. Stock allocation in general multi-echelon distribution systems with (r, s) order-up-to-policies. *International Journal of Production Economics*, 49(2):157–174, 1997. 2

[60] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016. 14

[61] Benjamin Van Roy, Dimitri P Bertsekas, Yuchun Lee, and John N Tsitsiklis. A neuro-dynamic programming approach to retailer inventory management. In *Proceedings of the 36th IEEE Conference on Decision and Control*, volume 4, pages 4052–4057. IEEE, 1997. ii, ii, 2, 3, 16, 17, 25, 36, 45, 57

[62] Tim van Tongeren, Uzay Kaymak, David Naso, and Eelco van Asperen. Q-learning in a competitive supply chain. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 1211–1216. IEEE, 2007. 12

[63] Nathalie Vanvuchelen, Joren Gijsbrechts, and Robert Boute. Use of proximal policy optimization for the joint replenishment problem. *Computers in Industry*, 119:103239, 2020. 67

[64] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015. 14

[65] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 15, 27

[66] Linwei Xin. Understanding the performance of capped base-stock policies in lost-sales inventory models. *Available at SSRN 3357241*, 2019. 8, 34

[67] Linwei Xin and David A Goldberg. Optimality gap of constant-order policies decays exponentially in the lead time for lost sales models. *Operations Research*, 64(6):1556–1565, 2016. 34

Appendix A

Hyper-parameters

Hyper-parameter	Value
Number of Layers	2
Number of Neurons	[64, 64]
Activation Function Policy Network	tanh
Activation Function Value Network	ReLU
Train Batch Size	327680
Mini-batch Size	32768
Number SGD Epochs	30
Number of workers	12
Learning Rate	10^{-4}
VF Loss Coefficient (c_1)	10^{-7}
Entropy Coefficient (c_2)	0
Gradient Clipping Norm	40.0
Discount factor γ	0.95
GAE λ	0.98

Table A.1: Hyper-parameters used for PPO training

- **Neural Network Architecture**: The neural network architecture did not affect the performance of the algorithm positively. Deeper (2, 3 or 4 layers) or wider (64, 128 or 256 units in a layer) network resulted in little to no performance gain. It seems like such a small network size is enough to form a very good policy.
- **Activation Functions**: The ReLU activation function for the value network has resulted in better value estimations compared to the tanh. This could be connected to the fact that the reward variations are rather high, and ReLU is more suitable for the task due to the wider activation range. Other activation functions such as ELU, SELU, or SWISH did not result in better estimates. The efficiency of activation functions were evaluated based on the explained variance metric, which equals $1 - \mathrm{Var}(\hat{V}_t - V_t^{target}) / \mathrm{Var}(V_t^{target})$. Surprisingly,

setting the hyperbolic tangent activation function on the policy network resulted in better convergence and more stable training compared with *ReLU*. It might be the case that the *tanh* works better in conjunction with the last *softmax* activation layer.

- **Training Batch Size and Mini-batch size**: Large training batch size increases sample complexity, however, results in more stable updates and faster wall-clock time training. A similar effect was noticed for the mini-batch size.
- **Number of SGD epochs**: Contrary to the results of [63], reducing the number of training epochs ($[1, 2, 5, 15]$) did not result in better convergence. In all of the experiments high number of epochs resulted in faster training and overall better performing policy.
- **Number of workers**: The number of workers only affects the speed of training since the network update is performed once the training batch is collected. More workers resulted in a faster collection of the batch size. The wall-clock training time improved dramatically with the number of workers: doubling the number of used logical cores resulted in half of the training time.
- **Learning rate**: Different learning rates, as well as learning rate schedules, did not lead to better performance.
- **Value function (VF) loss coefficient (c_1)**: The VF loss coefficient was chosen in a way that offsets huge negative rewards at the beginning of the training. In the case of a high VF loss coefficient, the value function was unable to converge.
- **Entropy coefficient (c_2)**: An increase in the entropy coefficient also did not result in better performance. We have tested different entropy coefficients $[10^{-3}, 10^{-5}, 10^{-7}, 10^{-9}]$. Any value apart from 0 resulted in slower convergence and worse cumulative rewards.
- **GAE λ**: The λ parameter for GAE was tuned on a number of settings starting from $\lambda = 0.9$ to $\lambda = 1.0$ with a step of 0.1. In most cases, values that were closer to one resulted in a good performance.

Appendix B

OWMR Scenario Costs

Table B.1 presents the average costs of the system under different policies across the ten scenarios for the one-warehouse multi-retailer problem. Next, the figures B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9, B.10 visualize the cost per category and the cost location for all scenarios presented in the main body.

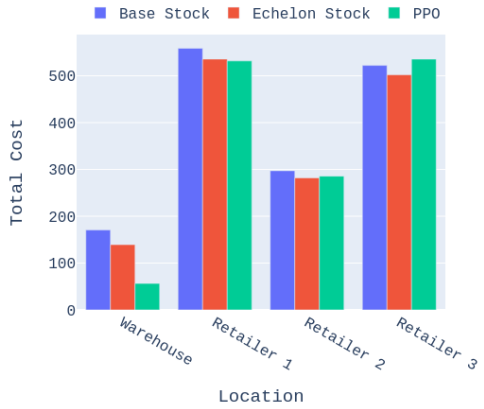
Scenario	Base Stock		Echelon Stock		PPO	
	Mean Costs	Std. Error	Mean Costs	Std. Error	Mean Costs	Std. Error
1	-1741.79	4.96	-1647.98	5.34	-1601.89	4.49
2	-1535.31	5.19	-1457.61	4.33	-1411.57	3.85
3	-1811.24	6.79	-1709.56	5.34	-1666.95	6.87
4	-1998.88	8.69	-1867.31	5.12	-1905.06	6.02
5	-2783.77	15.15	-2728.15	13.57	-2644.43	13.24
6	-1404.66	3.16	-1367.47	3.16	-1361.43	2.91
7	-1450.95	3.28	-1402.47	2.97	-1397.04	3.25
8	-1724.34	4.07	-1696.58	3.76	-1617.01	3.95
9	-1587.94	3.74	-1576.57	3.65	-1553.97	3.88
10	-2036.132	6.05	-2010.617	5.47	-1883.49	5.34

Table B.1: Average costs with corresponding standard error on OWMR scenarios

(a) Aggregate costs per location

(b) Costs per location.

Figure B.1: Costs for Scenario 1.



(a) Aggregate costs per location

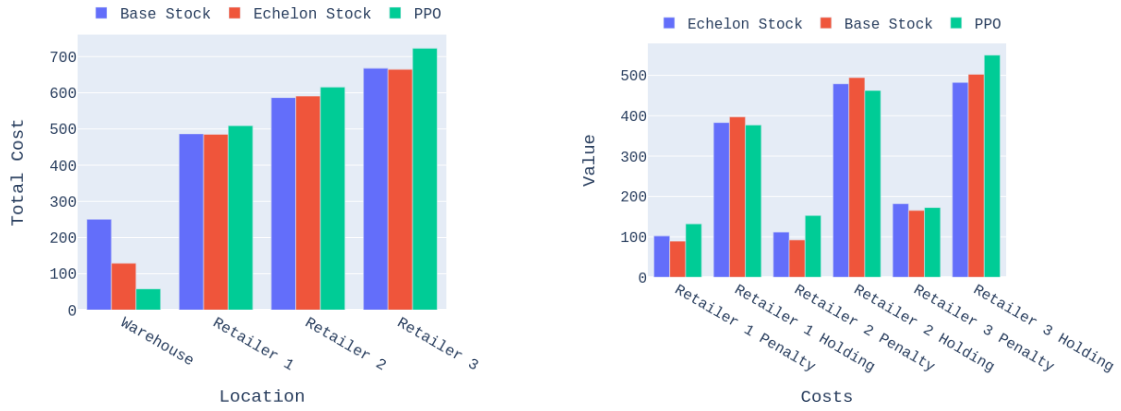
(b) Costs per location.

Figure B.2: Costs for Scenario 2.

(a) Aggregate costs per location

(b) Costs per location.

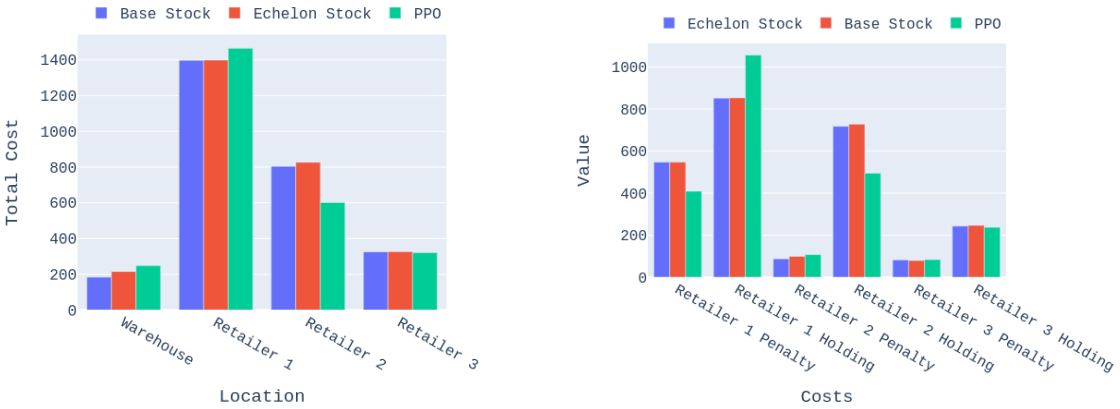
Figure B.3: Costs for Scenario 3.



(a) Aggregate costs per location

(b) Costs per location.

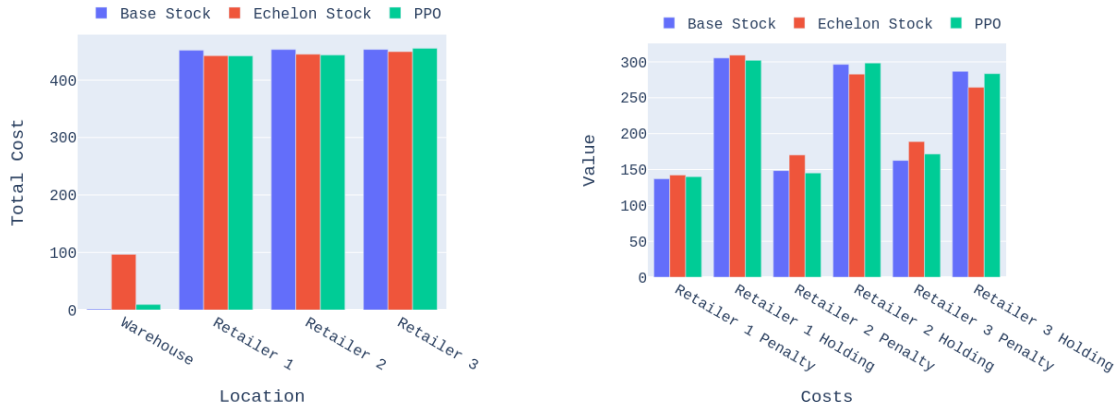
Figure B.4: Costs for Scenario 4.



(a) Aggregate costs per location

(b) Costs per location.

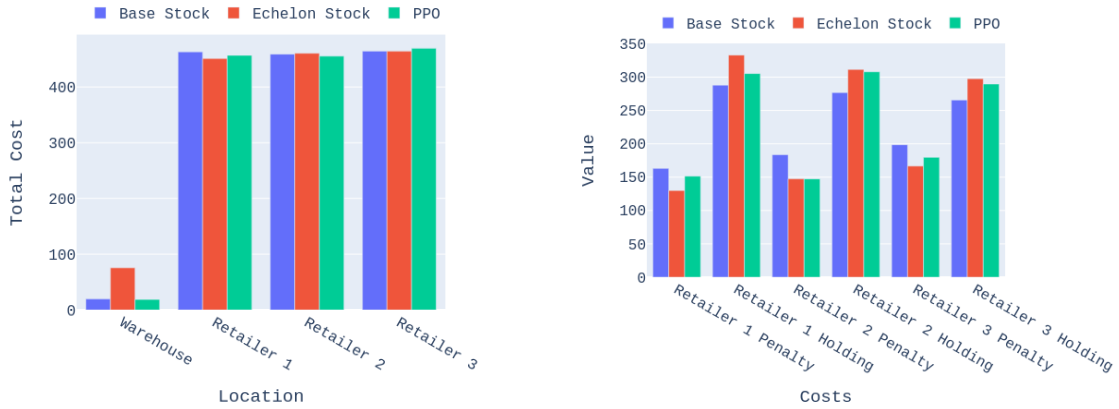
Figure B.5: Costs for Scenario 5.



(a) Aggregate costs per location

(b) Costs per location.

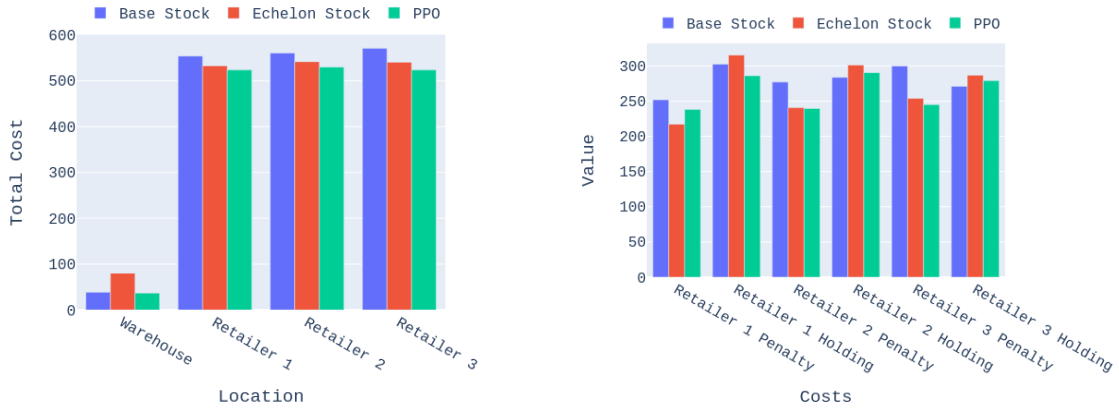
Figure B.6: Costs for Scenario 6.



(a) Aggregate costs per location

(b) Costs per location.

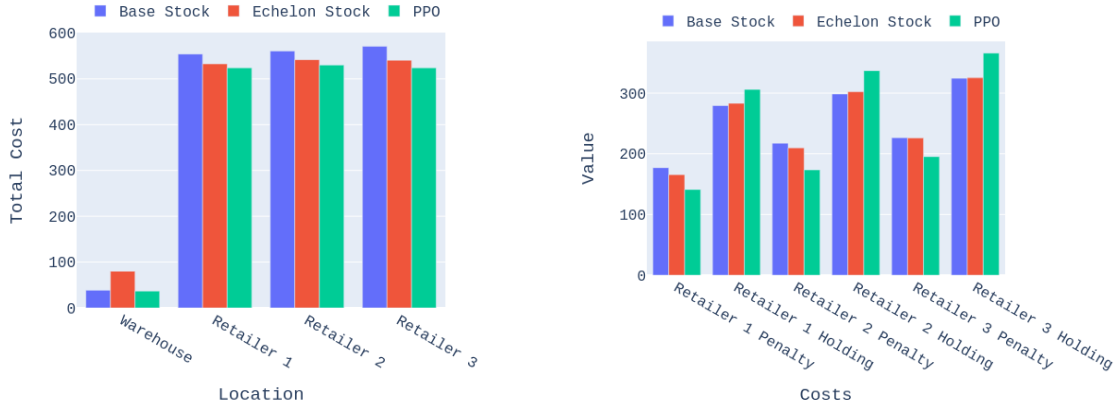
Figure B.7: Costs for Scenario 7.



(a) Aggregate costs per location

(b) Costs per location.

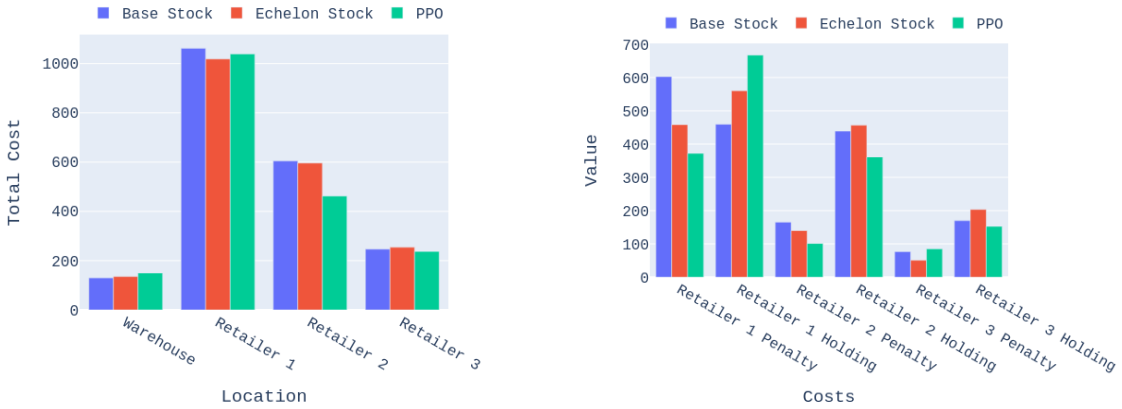
Figure B.8: Costs for Scenario 8.



(a) Aggregate costs per location

(b) Costs per location.

Figure B.9: Costs for Scenario 9.



(a) Aggregate costs per location

(b) Costs per location.

Figure B.10: Costs for Scenario 10.