Can Deep Reinforcement Learning Improve Inventory Management? Performance on Dual Sourcing, Lost Sales and Multi-Echelon Problems

Article *in* SSRN Electronic Journal · July 2019

DOI: 10.2139/ssrn.3302881

CITATIONS
13

READS
5,196

4 authors:

Joren Gijsbrechts
KU Leuven
6 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Robert Boute
KU Leuven
68 PUBLICATIONS **604** CITATIONS
SEE PROFILE

Dennis J. Zhang
Washington University in St. Louis
34 PUBLICATIONS **370** CITATIONS
SEE PROFILE

Jan Albert Van Mieghem
Northwestern University
125 PUBLICATIONS **3,480** CITATIONS
SEE PROFILE

Can Deep Reinforcement Learning Improve Inventory Management? Performance on Lost Sales, Dual Sourcing and Multi-Echelon Problems

Joren Gijsbrechts
Católica Lisbon School of Business & Economics, Portugal

Robert N. Boute
Vlerick Business School and KU Leuven, Belgium

Jan A. Van Mieghem
Kellogg School of Management, Northwestern University, United States

Dennis J. Zhang
Olin Business School, Washington University in St. Louis, United States

December 17, 2018; Revised Jul. 29, 2019; Oct. 6, 2020

Abstract. ***Problem definition:*** Is Deep Reinforcement Learning (DRL) effective at solving inventory problems? ***Academic/practical relevance:*** Given that DRL has successfully been applied in computer games and robotics, supply chain companies are interested in its potential in inventory management. We provide a rigorous performance evaluation of DRL in three classic and intractable inventory problems: lost sales, dual-sourcing and multi-echelon inventory management. ***Methodology:*** We model each inventory problem as a Markov Decision Process and apply the Asynchronous Advantage Actor Critic (A3C) DRL algorithm. We apply and tune the A3C algorithm in three classic inventory problems for a variety of parameter settings. ***Results:*** We demonstrate that the A3C algorithm can match performance of state-of-the-art heuristics and other approximate dynamic programming methods, with limited changes to the tuning parameters across all studied problems. Yet, the initial tuning remains computationally burdensome. Generating structural policy insight or designing specialized policies that are (ideally provably) near optimal thus remains desirable. ***Managerial implications:*** Our study provides evidence that DRL can effectively solve inventory problems. This is especially promising when problem-dependent heuristics are lacking.

Key words: Inventory Theory and Control, Logistics and Transportation, Supply Chain Management, OM-Information Technology Interface

1. Introduction

In the nexus of business and technology, no topic is hotter today than machine learning and artificial intelligence. We focus on deep reinforcement learning (DRL), the subfield of machine learning that develops "prescriptions" or policies for sequential decision-making problems. DRL employs deep neural nets to approximate the value or policy functions of Markov Decision Processes. This allows DRL to circumvent the curse of dimensionality, inherent to dynamic programming. With the help of DRL, programs have learned to play Atari games directly from image pixels [Mnih et al. 2015] and recently AlphaGo has beat the best human players in Go, the most complex board game in human history [Silver et al. 2016].

Despite the on-going frenzy about these breakthroughs, applications of DRL in industrial contexts such as inventory management remain rather scarce. We demonstrate how and when DRL can be applied to classic, yet intractable inventory problems. The true strength of these general learning algorithms is that they provide a way to solve a diversity of problems rather than relying on extensive domain knowledge or restrictive assumptions. In the inventory management literature a variety of model-specific heuristics exist; it typically depends on the model assumptions which heuristic performs best. In contrast, DRL algorithms are easily accessible and can be applied to any sequential decision-making problem. As such, DRL could be perceived as a general purpose technology that can serve many different purposes. We establish proof-of-concept with rigorous performance benchmarks to lost sales, dual-sourcing or dual-mode, and multi-echelon inventory models. Our conclusion is nuanced: DRL can match performance of state-of-the-art heuristics across a variety of problems, with limited changes to the tuning parameters. Yet, finding good initial hyperparameter sets is computationally burdensome while the resulting policies of DRL algorithms remain black boxes. Generating structural policy insight or designing specialized policies that are (ideally provably) near optimal thus remains a desirable research activity. At the same time, DRL provides promising results and has potential in practice, especially when problem-dependent heuristics are lacking.

We test the use of DRL on three classic inventory problems. In lost sales inventory models customers walk away or receive an expedited shipment at a premium cost when the desired product is not available (and hence no backlogging is allowed). Under dual sourcing, inventory can be replenished from a fast but expensive source and from a regular, cheaper source with longer lead time. In the equivalent dual-mode problem, inventory can be replenished from a single source using two complementary transportation modes. Multi-echelon inventory management includes additional stages or echelons of the supply chain that can hold inventory, such as an intermediate central warehouse between an upstream plant and downstream retailers.

These conceptually-simple problems have vexed supply chain management scientists for decades. Little is known about the optimal policy structure and traditional solution methodologies such as dynamic programming quickly become intractable due to the curse of dimensionality: their size grows exponentially as replenishment lead times get longer. Heuristic policies are typically problem-dependent and rely on restrictive assumptions, limiting their use in different settings. We explore whether a general purpose technology like DRL provides an alternative.

We aspire to make the following contributions: (1) Provide proof-of-concept that deep reinforcement learning can tackle a variety of inventory problems that have intractable dynamic programs; (2) Numerically evaluate performance relative to the optimal dynamic program and compare the

performance and ease-of-use versus heuristic policies and approximate dynamic programming methods; and (3) Provide a tutorial on how, why and when DRL works. Our aim is thus not to contribute to DRL per se; but to demonstrate how DRL may contribute to inventory management.

2. Selected Literature reviews

We provide a brief introduction to well-performing heuristics on lost sales (Section 2.1), dual sourcing (Section 2.2), and multi-echelon (Section 2.3) inventory models. These heuristics have been developed and shown to perform well under the conventional assumptions: i.i.d. demand, linear sourcing costs and deterministic lead times. As soon as one of these assumptions is relaxed or other practice-specific constraints are added, the performance of these heuristics may suffer and better performing policies are desired. This is precisely where numerical approximation methods may provide a solution, which we review in Section 2.4.

2.1. Lost Sales Inventory Models

Arrow and Karlin [1958] show that in a single sourcing backlogging model with constant lead time, inventory on hand and outstanding orders can be collapsed into one single number (i.e., the inventory position) and a single-dimensional base-stock policy is optimal. They also indicate that this state-space reduction no longer holds when excess demand is lost instead of backlogged. The optimal policy then depends on the entire inventory pipeline vector such that the complexity grows exponentially in the lead time. Morton [1969] provides more structural properties and bounds.

Base-stock policies generally perform poorly in lost sales inventory models [Zipkin 2008a], except for large penalty costs, in which case optimal inventory levels are high and stock-outs are rare. Huh et al. [2009] prove that base-stock policies are asymptotically optimal for large penalty costs. [Goldberg et al. 2016] show that the *Constant Order* policy with only constant replenishment from a single source [Reiman 2004] is asymptotically optimal for long lead times. Xin [2019] combines these two asymptotic results to propose the *Capped Base-Stock* policy.

The *Myopic* policy generally performs well for the lost sales inventory model [Morton 1971, Zipkin 2008a]: it computes in each period t the order quantity that minimizes the expected costs in the period when the order arrives (i.e. in period $t+l$, with l representing the lead time) by recursively computing the expected stock levels in period $t+l$. These findings are more rigorously confirmed by Brown and Smith [2014] who use the value of perfect information in combination with the L♮-convexity (read *el-natural*) property (which was proven by Zipkin [2008b]). They develop tight lower bounds and show how myopic policies consistently perform close to optimality on the lost sales problem even in settings where dynamic programming is intractable. Extensions such as the Myopic 2-period policy [Zipkin 2008a] perform even better. Further increasing the myopic horizon comes at the expense of increased computation time (similar to the dynamic program in which the

infinite horizon is used to choose actions). The one- and two-period myopic policies are essentially approximate dynamic programming methods as they require, at each time epoch, the iteration (or evaluation) over all states that can be reached during the considered horizon [Xin 2019]. They thus also suffer from the curse of dimensionality and are more computationally intensive than base-stock or constant order policies. Inspired by Zipkin's (2008b) finding that the value function of the lost sales problem is L♮-convex, Sun et al. [2016] develop a linear programming approximate dynamic programming (LP-ADP) algorithm. We elaborate further on LP-ADP in Section 2.4.

2.2. Dual Sourcing Inventory Models

Dual sourcing or dual-mode replenishment has been studied since the pioneering inventory management models of the mid-twentieth century. Fukuda [1964] proved that the optimal policy follows a base-stock structure when the lead time difference between both sources is exactly one period. When the lead time difference exceeds one period, however, a base-stock policy is no longer optimal [Sheopuri et al. 2010]. No simple structure prevails and the optimal policy depends on the vector of outstanding orders [Whittemore and Saunders 1977]. As costs related to replenishments within the expedite lead time do not impact the optimal policy [Sheopuri et al. 2010], the optimal policy depends on the $(l_r - l_e)$ outstanding orders, in which l_r and l_e respectively represent the lead time of the regular slow and expedite supplier. These insights inspired the development of various heuristic policies by collapsing the state vector to one or two inventory positions. For instance, the dual-base stock policies of Scheller-Wolf et al. [2003] and Veeraraghavan and Scheller-Wolf [2008] aggregate the pipeline inventory: the *Single Index (SI)* policy uses one inventory position over the long lead time while the *Dual Index (DI)* uses both the long and short lead time inventory positions. The *Capped Dual Index (CDI)* policy [Sun and Van Mieghem 2019] extends the dual index policy by introducing a cap on the order to the slow source. They prove that the CDI policy is robustly optimal, i.e., it minimizes the worst case performance across a range of deterministic demand scenarios. The *Tailored Base-Surge (TBS)* policy [Allon and Van Mieghem 2010] places a constant order to the slow source, and operates with a base-stock policy to determine orders at the fast source. Xin and Goldberg [2017] prove that the TBS policy is asymptotically optimal when the lead time difference between both sources grows to infinity. Sheopuri et al. [2010] show that dual sourcing is a generalization of lost sales inventory management. This is also apparent in the type of heuristic policies for both models. The *Constant Order* policy [Reiman 2004], for instance, is a Tailored Base-Surge policy with only constant replenishment from a single source, that is asymptotically optimal for long lead times in lost sales inventory models [Goldberg et al. 2016]. The (Capped) Dual Index policy in dual sourcing adds an additional base-stock level to the lost sales (Capped) Base-Stock policy.

Sheopuri et al. [2010] propose policies making better use of the structure of the pipeline inventory vector: the *Weighted Dual Index* policy weighs different elements of the inventory vector prior to ordering up to the base-stock levels while *Vector Base-Stock (VBS)* policies employ a base-stock for the expedited source and a vector base-stock policy identical to Zipkin [2008a] for the slow source. Hua et al. [2015] show that the dual sourcing value function satisfies L♮-convexity, similar to the lost sales inventory model [Zipkin 2008b]. Inspired by this property, they show that the optimal orders to the slow source are more sensitive to the longest outstanding orders, and the optimal orders to the fast source depend more on the soon-to-arrive outstanding orders. By developing upper and lower bounds on the slow source's order and using a weighted average of these bounds to determine the slow source's orders, they generalize the VBS policy to the *Best Weighted Bounds* policy. Chen and Yang [2019] exploit L♮-convexity to develop an LP-ADP algorithm to dual sourcing with supply uncertainty, identical to that of Sun et al. [2016] for lost sales inventory models.

2.3. Multi-Echelon Inventory Models

Multi-echelon models include multiple stages or echelons in the supply chain that can hold inventory. For instance, central warehouses can pool inventory prior to allocating it to downstream retailers. Seminal works such as Clark and Scarf [1960] and Federgruen and Zipkin [1984] characterize the optimal policy structure under several strong assumptions such as, respectively, having a serial system (i.e., no parallel stocking facilities) or no option to stock at intermediate terminals. de Kok et al. [2018] provide an extensive review of the variety of multi-echelon inventory models studied based on number of echelons, network structure, holding cost functions, etc. They also mention that there is little hope to find the optimal policy structure in divergent multi-echelon systems, such as the one we investigate.

We study the multi-echelon inventory model employed in Van Roy et al. [1997]. They adopt a neuro-dynamic programming approach to solve a two-echelon model with one warehouse and multiple retailers. The model is a hybrid between backlogging and lost sales, as described in Section 7. Closely related to this model are the partial lost sales and divergent network models studied in Nahmias and Smith [1993, 1994].

2.4. Approximate Dynamic Programming

Markov Decision Processes (MDPs) provide a mathematical framework to solve sequential decision-making problems with countable state and control spaces [Puterman 1994]. Traditional solution methodologies such as linear programming (LP) or dynamic programming (DP) using either value or policy iteration, generally do not scale well to large problem sizes. Therefore, several *approximate dynamic programming* (ADP) methods have been developed as described in the seminal book

by Powell [2011]. We provide a selective review, starting from methods that heavily exploit the problem structure to the most general purpose technologies available to date.

One branch of ADP methods exploits the problem structure. For example, for inventory models with zero or one period lead time, Halman et al. [2009] and Chen et al. [2014] avoid the need to iterate over the entire state space, speeding up the DP optimization and allowing the algorithm to solve close to polynomial time. Because of their heavy reliance on the problem structure, Chen et al. [2014] discuss the desire of more general purpose models. Our work may address this desire.

A second branch of ADP methods reduces the size of the problem by aggregating states, e.g. by clustering states based on features. Fang et al. [2013] and Giannoccaro and Pontrandolfo [2002] cluster the pipeline inventory into subsets by aggregating parts of the pipeline inventory. Based on their expertise into the specific problem, Van Roy et al. [1997] manually select 23 features from a multi-echelon supply chain to cluster the state space. These features include aggregations of parts of the inventory vectors, the variance among retailer inventory levels, products of inventory levels etc. It is often left up to the discretion of the user how to choose good cluster sizes, which can be a formidable challenge. Keller et al. [2006] automate this state aggregation process by using a neighborhood component analysis. While state aggregation can speed up computation, important information can get lost. This is always the case when the optimal policy depends on each element of the state, as in the three inventory problems that we study.

A third branch approximates the value or policy functions of MDPs to generate near-optimal policies. Selecting a good approximation is challenging if little is known about the structure of the optimal value or policy function. In inventory management, various function approximations are used, such as a linear combination of the state variables [Keller et al. 2006] or fitting a specific quadratic function on the state variables [as in Sun et al. 2016, Chen and Yang 2019]. The latter exploits the L♮-convexity property of respectively the lost sales and dual sourcing model with supply uncertainty to fit a specific L♮-convex quadratic shape, as introduced by Murota [2003]. As it is often infeasible to explore the entire state space when approximating the value function, efficient sampling of states is required. Some methods rely on well-known heuristics to sample states: Sun et al. [2016] use the myopic lost sales policy to sample states and Chen and Yang [2019] leverage the Single/Dual Index and Tailored Base-Surge policies. Both then apply linear programming to fit the approximation coefficients of the value functions using the sampled states.

Linear programming approximate dynamic programming (LP-ADP) falls in the third branch of ADP techniques. It assumes a specific functional approximation of the sample states using a well-known heuristic and uses linear programming to obtain the coefficients of these value functions. This approach circumvents the computational explosion characterizing traditional linear programming to solve MDPs: the need to include the value function of each state in the objective while each

state-action pair requires a unique constraint. Instead, only the sampled states are included in the objective function and only the sampled state-action pairs are included in the constraints. Solving the LP then provides the coefficients of the approximating value functions, which then provides a vehicle to evaluate the value function at unvisited states. An "LP-greedy" policy minimizes the current cost plus the cost-to-go of the next stage. Both Sun et al. [2016] and Chen and Yang [2019] report small savings in comparison to the sampling policies. We employ LP-ADP as a benchmark in our numerical experiments.

Reinforcement learning (RL) is a mathematical framework to solve MDPs without requiring an exact representation of the environment. An RL agent *learns* to maximize expected rewards by interacting with an environment. Mathematically, RL algorithms develop a good approximation of the value or policy function of the underlying MDP. Choosing a good functional approximation and efficient sampling of states and actions is key. While there are RL algorithms that heavily exploit problem structure, we focus on the recently developed general purpose RL algorithms. A major benefit of these RL algorithms in comparison to problem-specific heuristics is that they can *learn* policies in environments with less stylized or non-conventional assumptions. This is valuable in settings where off-the-shelf inventory replenishment heuristics are not available or do not perform well. A well-performing generic technology then adds value compared to tailor-made policies for one specific company or business.

Although classic RL algorithms, such as temporal difference and Q-learning, had some success in the past [Tesauro 1995, Kohl and Stone 2004, Watkins 1989], original RL approaches still suffered from the lack of scalability [Strehl et al. 2006]. Therefore, in recent years, reinforcement learning is combined with deep learning i.e., powerful function approximation using deep neural networks. Deep reinforcement learning (DRL) thus uses deep neural networks to approximate the value or policy functions of MDPs. Although both RL and DRL have been studied extensively in the Computer Science and Operations Research literature [Sutton and Barto 1998, Mnih et al. 2015], the accessibility of more computational power and recent algorithmic breakthroughs have sparked new interest in DRL. The excellent performance of DRL in a set of Atari games reported in Mnih et al. [2015] was an important milestone. By embedding target networks and experience replay in deep Q-learning, the performance of DRL algorithms peaked. These algorithmic improvements add an additional neural net and a large history of observations, respectively, which stabilizes learning. It inspired many follow-up papers improving the performance of deep Q-learning further. Notable extensions include the use of dueling networks [Wang et al. 2015], double Q-learning [van Hasselt et al. 2015], prioritized experience replay [Schaul et al. 2015], or the rainbow implementation of Hessel et al. [2017] that combines and compares these extensions.

Unlike Q-learning methods that approximate the value functions of the underlying MDP, policy-based methods directly develop a policy. The REINFORCE algorithm of Williams [1992], for instance, uses gradient ascent to develop a stochastic policy. Since the algorithm learns on-policy (i.e., it only uses the most recent observations to update the policy) the samples are correlated and non-stationary. This causes learning to be less stable as the policy may over-fit on certain regions of the state-space. The Asynchronous Advantage Actor-Critic (A3C) algorithm that we will employ uses parallel learners and a critic as a baseline to improve the speed and stability of the learning process. We will elaborate further on this actor-critic approach in Section 3. When we started this research project A3C was one of the most popular DRL algorithms due to its excellent performance and fast training time [Mnih et al. 2016]. We show that with only minor modifications, it is capable to develop good policies on three intractable inventory problems. Yet, as our numerical experiments also confirm, A3C remains rather sensitive to the employed hyperparameters and not all models converge to good policies. Follow-up papers have tackled these limitations. Notable breakthroughs include trust region methods, such as Trust Region Policy Optimization and Proximal Policy Optimization [Schulman et al. 2017a,b], that prevent the policy update from being too large. These models are less sensitive to the hyperparameters while training is more stable and convergent.

A vast literature exists on optimizing the hyperparameters of machine learning algorithms. We adopt a random search approach (see also Section 4). Bergstra and Bengio [2012] show (both empirically and theoretically) that random search outperforms naive grid search, without being more complex to implement. In random search samples are randomly drawn from specified intervals; as such, it is arguable even more easy to implement than a grid search, in which the modeller specifies a grid of potential values from which the hyperparameters are picked. Bergstra et al. [2013] demonstrate how random search may be improved upon by using more complex tuning approaches. Bayesian models, for instance, rely on a probability model to estimate how different hyperparameter combinations will perform. We opted for a random search approach as it has several key benefits compared to more complex methods, as also outlined by Bergstra and Bengio [2012] and Bergstra et al. [2015]: (1) *simplicity:* random search only requires the modeller to define the interval on which one should sample without building an additional optimization framework; (2) *parallelization:* Bayesian models require observations (i.e., outcomes of the algorithm) to be able to estimate the performance of each hyperparameter (i.e., to develop the surrogate objective). This requires at least some runs to be made sequentially to perform the Bayesian update. In random search all runs can be parallelized; (3) *unbiased analysis:* random search generates independent and identically distributed samples. The resulting unbiased samples facilitate visualization of the results and avoid over-fitting on local minima.

Supervised machine learning models thrive on the availability of more data than ever, elevating their performance to higher levels. The relation between data and the performance of RL algorithms is more nuanced. DRL algorithms are used to generate policies for complex problems without requiring the availability of data; all that is needed is an environment to interact with. For instance, while Alpha Go learned to play the game of Go by using data of games of human experts [Silver et al. 2016], its update Alpha Zero learned from scratch [Silver et al. 2017] solely by playing the game against itself, effectively generating data through self-play. Even though DRL algorithms may require millions of training iterations, they do not require excessive amounts of training data, as long as a model of the environment exists that generates the random variables of the problem. Evidently, the model (or simulation engine) may need data to keep generating the random variables. In many practical settings the distribution of the uncertainty is unknown and must be derived from historical samples. We refer to Levi et al. [2015] for an excellent discussion on the impact of using empirical samples rather than the true distribution on the classic Newsvendor problem.

There are only limited applications of DRL to inventory management. Oroojlooyjadid et al. [2017] applied deep Q-learning to the Beer Distribution Game. While our paper also applies DRL to inventory problems, it differs from this working paper in two distinctive ways. First, we provide optimality gaps on three different inventory problems whose stochastic dynamic program become quickly intractable. Second, we show the versatility of DRL by applying it to three different inventory problems with limited modification of the algorithm, thereby demonstrating that DRL resembles a general purpose technology.

3. DRL Solution Approach to Lost Sales Inventory Replenishment

Deep Reinforcement Learning provides a way to approximate and solve large Markov Decision Processes for which traditional dynamic programming methods are intractable. We will demonstrate in this section how the Asynchronous Advantage Actor-Critic (A3C) algorithm operates. We use the lost sales inventory model as our focal problem and show in later sections how dual sourcing and multi-echelon problems can be plugged into the same framework, showing the versatility of the algorithm.

Consider the periodic-review inventory replenishment of a single item over an infinite horizon. In the conventional lost sales model, inventory can be replenished at unit cost c with lead time l. At the beginning of any period t, the order quantity, $q_t \geq 0$, must be decided knowing the last observed inventory on hand, I_{t-1}, and outstanding receipts or "pipeline" vector $Q_{t-1} = (q_{t-l}, q_{t-l+1}, \cdots, q_{t-1})$. We note that this formulation assumes strictly positive lead times. If $l = 0$, there are no outstanding orders as the order $q_{t-l} = q_t$ is immediately received and added to the on-hand inventory. Finally, the unknown demand d_t is realized and subtracted from the on-hand inventory. Excess demand is lost so that inventory evolves as $I_t = [I_{t-1} + q_{t-l} - d_t]^+$ and the pipeline vector as $Q_t = (q_{t-l+1}, \cdots, q_t)$.

The lost sales problem can be modeled as a Markov Decision Process with state $S_t = (I_{t-1}, Q_{t-1})$ at time t. As orders placed in period $t-l$ are available at the beginning of period t, they can be added to the ending inventory of period $t-1$ such that the state vector becomes l-dimensional: $S_t = (s_0 = I_{t-1} + q_{t-l}, s_1 = q_{t-l+1}, s_2 = q_{t-l+2}, \cdots, s_{l-1} = q_{t-1})$. Let $\mathcal{S}_t$ denote the set of admissible states at time t. The action vector for the lost sales problem is one-dimensional consisting of the units ordered $a_t = (q_t)$. We opt for a vector notation to accommodate for the multi-dimensional action spaces in dual sourcing and multi-echelon models in Sections 6 and 7, respectively. After taking action a_t, the cost c_t incurred in period t consists of sourcing costs, per-period holding cost h for each unit held in inventory-on-hand or per-period shortage cost p per unit of lost demand:

$$c_t(S_t, a_t) = q_t c + h[I_t]^+ + p[d_t - I_{t-1} - q_{t-l}]^+. \tag{1}$$

Let $\mathcal{A}_t$ denote the set of admissible order policies $\pi_t = \{a_{t+j} : j = 0, 1, \cdots\}$. The objective is to find an admissible order policy that minimizes the expected present value $\mathcal{C}_t$ of future costs. Assuming henceforth sufficient regularity including countable, compact state and action spaces, this cost when starting from state S_t and following policy π_t is:

$$\mathcal{C}_t^{\pi_t}(S_t) = \sum_{j=0}^{\infty} \gamma^j \mathbb{E}^{\pi_t} c_{t+j}(S_{t+j}, a_{t+j})$$

where $\gamma \in (0,1)$ is the discount factor and $\mathbb{E}^{\pi_t}$ is the expectation operator when following policy π_t. Define the optimal value function $v(S_t)$ as the infimum of $\mathcal{C}_t^{\pi_t}(S_t)$. Assuming sufficient regularity and countable states and actions, the value functions are obtained when following an optimal policy by solving the celebrated recursive *Bellman equations* [Bellman 1954]:

$$v(S_t) = \min_{a_t \in \mathcal{A}_t} \left\{ c_t(S_t, a_t) + \gamma \sum_{S' \in \mathcal{S}_{t+1}} \mathbb{P}(S_{t+1} = S' | S_t, a_t) v_{t+1}(S') \right\}.$$

Despite their simple appearance, solving these Bellman equations can quickly become problematic as their problem complexity suffers from the triple curse of dimensionality [Powell 2011], which is driven by the dimension of (1) the l-dimensional state space; (2) the uni-dimensional action space; and (3) the transition probability matrix, dependent on the size and number of possible demand realizations. Note that we assume integer or discrete demand; continuous only makes it harder.

To cope with the curse of dimensionality, the dimensions of the problem can be reduced by applying approximate dynamic programming. As discussed in Section 2.4, one ADP branch includes *reinforcement learning*. In their simplest form, value-based methods such as tabular Q-learning [Watkins 1989] store estimated values of $v(S)$ in lookup tables. In Q-learning, *action-value functions* $Q(S_t, a)$ represent the future expected cost of taking action a from state S_t and then following the

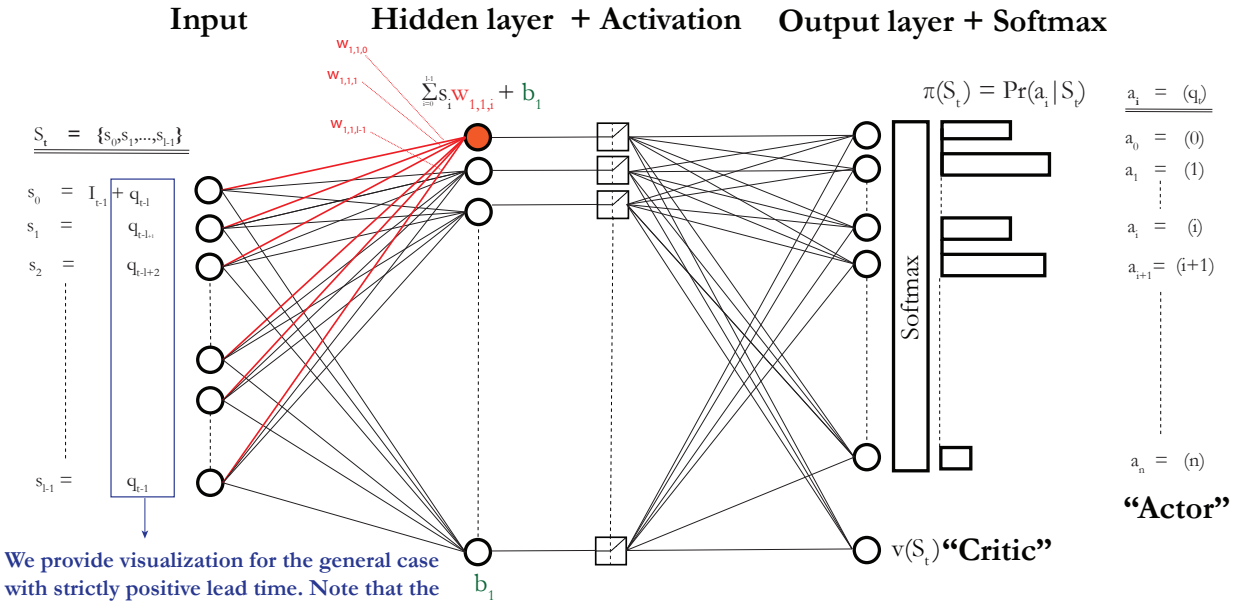

Figure 1 **Visualization of a simple neural net of the A3C where the input dimension is l, one hidden layer is used and n actions can be chosen. The output consists of (1) a value $v(S_t)$ (estimated by the "critic") when using (2) a stochastic policy over the action space $\pi(S_t) = \mathbb{P}(a_t|S_t)$ (estimated by the actor using a softmax function on the last layer that normalizes all values into a probability vector).**

best-known policy from state S_{t+1}. Note that the Q-values only differ in the first step from the value function $v(S)$. These Q-values can iteratively be improved: $Q_{t+1}(S_t, a_t) = (1-\alpha)Q_t(S_t, a_t) + \alpha(c_t + \gamma \max_{a\in\mathcal{A}} \mathbb{E}Q(S_{t+1}, a))$, in which α represents the learning rate that trades off how much the new observation $c_t + \gamma \max_{a\in\mathcal{A}} \mathbb{E}Q(S_{t+1}, a)$ is used to update the initial estimate $Q_t(S_t, a_t)$. New states and actions can be explored by following ϵ-greedy methods that trade off following the best-known policy, with probability $(1-\epsilon)$ against exploring new actions, with probability ϵ.

More sophisticated value-based methods use parametric approximations of the values $v(S)$ to circumvent the curse of dimensionality when storing all function values in lookup tables. Neural networks are an example of approximators of non-linear functions. Neural nets have an input layer, several hidden and activation layers in sequence, and an output layer. Figure 1 illustrates a neural net with one hidden layer. Its input is the $l-$dimensional state vector $S_t = (s_0 = I_{t-1} + q_{t-l}, s_1 = q_{t-l+1}, s_2 = q_{t-l+2}, \cdots, s_{l-1} = q_{t-1})$. A layer's output is a linear function of its input followed by a non-linear "activation layer." The value of the first node of the hidden layer is $\sum_{i=0}^{l-1} w_{1,1,i} s_i + b_1$ where the weights of the red edges $(w_{1,1,1} \cdots w_{1,1,l})$ and the bias b_1 are tuning parameters. The activation layer adds non-linearity to the linear function. For example, a positive-part operator, called "rectified linear unit" ReLU, only passes positive node values to the next layer. Concatenating

layers provides powerful approximations to non-linear value functions. Traditional neural networks use mostly one or two layers while recent *deep learning* employs a larger number of hidden layers.

In contrast to value-based methods, *policy-based* methods directly develop a policy. A deterministic policy prescribes for each state a single action. A stochastic policy prescribes for each state a probability distribution over all actions that specifies the probability that each action should be taken. At the juncture of value-based and policy-based methods, actor-critic methods combine both techniques by relying on an *actor* who develops a policy that is evaluated by a *critic*. We will explore the power of one such actor-critic method, namely the Asynchronous Advantage Actor-Critic (A3C) algorithm. We note that even though the transition probabilities are *known* by the MDP, the A3C algorithm does not have direct access to these probabilities but needs to learn them through interaction with its environment. In the RL literature this is referred to as model-free learning. The alternative approach (which we do not employ) is to use model-based algorithms such that the DRL algorithm has direct access to the transition function of the MDP. Model-based models may for instance use Monte Carlo Tree Search to look ahead and explore good actions.

In what follows, we use notation θ to define the set of all parameters that must be fine-tuned during training of the model, i.e., all weights of the edges and biases of the global neural net (θ^g) and of the local nets (θ^i) of each parallel learner i. (We describe the use of the global and local nets in the next paragraph.) Given set θ^g, A3C generates for every state S_t a stochastic policy: $\pi(S_t;\theta^g) = \mathbb{P}(a_t|S_t)$ (i.e., a probability distribution over all actions) and one value function $v^\pi(S_t;\theta^g)$ (representing the expected future discounted cost of following the stochastic policy). In our experiments, we use a fully connected network, implying that all nodes between layers are connected and the actor and critic use the same hidden layers. This delivered good performance on our inventory problems, but it does not necessarily need to be the case. The output of the actor passes a "softmax" function that normalizes the output layer values into a proper probability vector that defines the stochastic policy, as shown in Figure 1.

The A3C algorithm employs a special structure where n parallel learners each have an individual neural net with parameter set θ^i, $\forall i \in \{1, \cdots, n\}$. Together, yet asynchronously, the learners update the global neural net with parameter set θ^g. The total set of trainable parameters of the model θ thus includes both θ^i, $\forall i \in \{1, \cdots, n\}$ and θ^g. (We will later show that we use the local net with parameter set θ^i to compute the gradient of the loss function but apply the update on the global net with parameter set θ^g.) Each learner i interacts independently with its own copy of the environment by using its own neural net. The agents learn and update the global network through a training *buffer* of m periods of observed states, actions and costs as follows: During the k-th buffer, the current policy π_k is simulated for m subsequent periods starting at state $S_{m(k-1)}$. During the simulation, each parallel learner keeps track of the incurred costs $(c_{m(k-1)}, \cdots, c_{mk})$,

the actions taken $(a_{m(k-1)}, \cdots, a_{mk})$, the visited states $(S_{m(k-1)}, \cdots, S_{mk})$ and the resulting state at the end of this episode (S_{mk+1}). Given these records, each learner computes its loss function for training iteration and buffer k, denoted by Loss_k and to be defined in detail soon. Each learner i then computes the gradients of the total loss with respect to the parameters θ^i of its own local neural net and iteratively adjusts the parameters of the global net θ^g using a stepsize $\alpha > 0$ in the direction of the gradient to reduce the loss:

$$\theta^g_{k+1} = \theta^g_k - \alpha \nabla_{\theta^i} \text{Loss}_k / ||\nabla_{\theta^i} \text{Loss}_k||.$$

This process happens *asynchronously*, meaning that updates to the weights of the global network are made immediately whenever an agent has simulated its buffer and computed its gradient. The weights of the global network are thus continuously updated by using the gradients from the local learners. Once the global network is updated, the weights of the global network are then copied back to the local learner. The motivation for using parallel learners is that each learner is interacting with its own copy of the environment, which reduces the probability of over-fitting on specific state regions. To avoid exploding gradients, a clipping function bounds the gradients. The step size depends on the used optimization method. We employ Adam, developed by Kingma and Ba [2015], which dynamically adapts the step size based on the past updates.

The total loss function used is the sum of three specific losses that we now define:

$$\text{Loss} = \text{Value Loss} + \text{Policy Loss} + \text{Entropy Loss}.$$

The loss function and its gradient are computed for each parallel learner i and applied asynchronously to the global net. The value loss measures the quality of the value function approximation by the difference between the future discounted cost and the approximated value function at each observed state in the episode buffer. The future discounted cost of a state $S_p = S_{mt-k}$ in the t-th episode buffer consists of the costs observed k steps until the end of the episode buffer, $C^{(k)}_p = \sum_{i=0}^{k} \gamma^i c_{p+i}$, plus the infinite-horizon discounted costs after the episode buffer, which is approximated by the value function in the last period of the buffer, $\gamma^{k+1} v^{\pi_t}(S_{tm+1}; \theta^i)$. The value loss is then defined as the sum of the squared errors of all states $(S_{m(t-1)}, \cdots, S_{mt})$ within the episode buffer. We use a value regularization term β_V (which we fixed at 0.25) to prevent over-fitting:

$$\text{Value Loss} = \beta_V \sum_{p=0}^{m} \left(-v^{\pi_t}(S_{m(t-1)+p}; \theta^i) + C^{(m-p)}_{m(t-1)+p} + \gamma^{m-p} v^{\pi_t}(S_{mt+1}; \theta^i) \right)^2.$$

The policy loss is used to improve the policy by selecting "better" actions. For each action taken at period p during the episode t, we compute the differences between the cost of this action plus the discounted value of the next state $(c_t(S_t, a_t) + \gamma v^{\pi_t}(S_{t+1}))$ and the value function at the current

state ($v^{\pi_t}(S_t)$). This difference can be positive, which means that the action is not optimal using the current value function, zero, or negative, which means the current action is better than the action suggested by the policy. Therefore, minimizing this difference helps to choose better actions with respect to the current value function. We use Generalized Advantage Estimation [Schulman et al. 2015], in which the observed differences within the episode buffer are once again discounted. Generalized advantage estimation uses an additional weighing parameter $\lambda \in [0,1]$ which we keep fixed at 1 as it provides good performance in our setting. Since, at a given state, each action is only taken with a certain probability, the loss is corrected by multiplying by the log of the probability of selecting action a_t:

$$\text{Policy Loss} = \sum_{i=0}^{m} \left(\log \mathbb{P}(a_{m(t-1)+i}|S_{m(t-1)+i}) \sum_{p=0}^{k} \gamma^p \left(c_{m(t-1)+i} + \gamma v^{\pi_t}(S_{m(t-1)+i+1};\theta^i) - v^{\pi_t}(S_{m(t-1)+i};\theta^i) \right) \right).$$

Finally, the entropy loss is used to avoid the A3C algorithm over-fitting and converging to local optima. Minimizing entropy ensures that the model keeps the right balance between exploring new actions and exploiting actions that are known to perform well. The entropy for each episode buffer is defined as the entropy of the probability function over actions taken, which is the logarithm of the probability mass function over actions taken: $\mathbb{P}(a_i|s_i)\log\mathbb{P}(a_i|S_i)$. This entropy is minimized when all actions have the same probability and is equivalent to encouraging the algorithm to explore new actions. An entropy regularization term β_E determines how much exploration is added to the loss function:

$$\text{Entropy Loss} = \beta_E \sum_{p=0}^{m} \mathbb{P}(a_{m(t-1)+p}|S_{m(t-1)+p}) \log \mathbb{P}(a_{m(t-1)+p}|S_{m(t-1)+p}).$$

4. How to tune a DRL algorithm?

Thanks to the open nature of the machine learning community, the code for many deep reinforcement learning algorithms is freely available. While this should greatly facilitate their application, it is our experience that the cumbersome tuning process may demotivate researchers to explore the potential of DRL methods. Indeed, determining the suitable hyperparameters of the A3C algorithm (summarized in Table 1) is a non-trivial and very time-consuming task because evaluating hyperparameter sets is expensive and noisy. Therefore, we share our experience on how to achieve good performance for three different inventory problems, together with our code, which will be shared along with the publication of this paper. This may help other researchers to directly build on our experience when applying DRL to other operations problems.

For our initial models, we tested various hyperparameter settings and kept track of well-performing hyperparameter values. To further improve the results, we stored the best models (i.e., trained neural networks) and restarted from these models using different entropy factors and

learning rates. This further improves already well-performing models and uses less exploration and more exploitation as training progresses. This labor intensive tuning process motivated us to develop an automated tuning strategy without manual intervention, which we also tested to compute optimality gaps for the lost sales and dual sourcing problems. In addition, we selected a set of well-performing hyperparameter values that we used for a more extensive sensitivity analysis (in Section 5.2).

Building on our experience of manually tuning various model instances, we developed the following automated tuning process for the A3C algorithm: First, we decided to keep the following hyperparameters fixed: four layers in the neural network with widths 150, 120, 80 and 20, respectively; each layer followed by a ReLU activation; value regularization 0.25; four parallel learners; and clipping rate 40. These hyperparameters did perform well and any manual tuning deviations did not result in any significant improvement. Equivalently, the computational effort to further optimize those parameters did not outweigh their benefits in our experience. Therefore, they provide a good starting point. Of course, it is possible that changes to these hyperparameters can perform equally well or better in other settings.

Second, we took 200 random sample points of the remaining three hyperparameters—i.e., the length of the buffer, the entropy regularization rate and the learning rate—over pre-defined ranges. While the length of the episode buffer is simply sampled on a linear scale, both the entropy regularization rate and the learning rate are sampled on a log scale. This means that for a tuning range $[10^{-a}, 10^{-b}]$, we sample x on the uniform interval $[a, b]$ and set our hyperparameter to 10^{-x} . We determined those ranges by monitoring the three components of the loss function (value, policy and entropy loss) during manual training. This provided insight into the relevant ranges of the three parameters where the A3C algorithm tended to converge to good results. We evaluated the expected cost of the A3C policy by simulating 10 sample paths of 100,000 periods. We note that to further reduce the confidence intervals of all results in this manuscript, we additionally simulated all policies developed by the A3C algorithm on 100 sample paths of 100,000 periods.

Third is the final choice of the three hyperparameter values. The simplest solution is to pick the sample point with the best results. We went one step further and fitted a quadratic function, specifically a 3D ellipsoid, through the lower convex hull of the 200 points to highlight a well-performing region of the tuning parameters. Later, we will provide a visualization of this approach. We used an ellipsoidal convex hull merely to indicate a good parameter region; while its minimum may suggest a well-performing parameter at this point it is just that: a suggestion. More research on the applicability of an ellipsoidal convex hull seems desirable but is outside the scope of this paper.

Hyperparameter	**Well-performing values**	**Range for tuning**
Number of layers	4	
Width of layers	[150,120,80,20]	
Value regularization (β_V)	0.25	
Activation functions	ReLU	
Number of parallel learners	4	
Clipping rate	40	
Learning rate ($\alpha = 10^x$)	$x = -4$	$x \in [-7, -2]$
Entropy regularization ($\beta_E = 10^x$)	$x = -7$	$x \in [-10, -2]$
Buffer size	20	[20,100]

Table 1 Hyperparameters of the A3C algorithm that require tuning. The first column contains one set that performs well across all settings. The second column contains the ranges used in the automatic tuning process.

Table 1 summarizes both the values of hyperparameters that are kept fixed as well as the ranges used in the automatic tuning of the remaining three hyperparameters. Note that the learning rate and entropy regularization are sampled from a continuous interval, while the buffer size is constrained to be integer. Once the hyperparameters are chosen, the four agents start training.

Performance of the agents during training is evaluated every 100,000 periods by computing the average cost per period on a fixed sample path of 100,000 periods. When costs do not improve during 30 consecutive evaluations, the agent stops training. This is equivalent to a simulation of 3 million periods, with an intermediate cost evaluation every 100,000 periods. Once all agents are finished, a new set of hyperparameters is chosen. The evaluation of one set of hyperparameters took us about 24 CPU hours. All models were trained using (university) cloud servers. For each of the settings reported in Sections 5.1 and 6 (optimality gaps of lost sales and dual sourcing, respectively) we evaluated the A3C algorithm on approximately 250 hyperparameter settings (including both the manual search to set the ranges and the automatic tuning). The initial tuning of the dual sourcing and lost sales model thus required about 3000 of these "runs." This resulted in total training costs of around $2500 at current Google Cloud Platform rates of $0.033174 per virtual CPU hour. For the sensitivity analysis (Section 5.2) we ran the algorithm for approximately 2000 additional runs, resulting in an additional cost of around $1500. These large numbers contrast sharply with the computational effort necessary for our benchmarks. The benchmark heuristics typically have 1 to 3 policy parameters and optimization takes seconds or minutes. The methods that rely on approximate dynamic programming (the myopic policy and LP-ADP algorithm) partly circumvent the curse of dimensionality; as such, their running time can typically be expressed in minutes or hours rather than in days or weeks as is the case for the A3C algorithm.

5. Performance Evaluation of DRL in Lost Sales Inventory Models

To allow for a stringent performance analysis of the A3C algorithm in inventory management we develop two sets of experiments. The first experiment considers small-scale settings for which the

Hyperparameter	Tuning Range	Setting 1 (p=4, l=2)	Setting 2 (p=4, l=3)	Setting 3 (p=4, l=4)	Setting 4 (p=9, l=2)	Setting 5 (p=9, l=3)	Setting 6 (p=9, l=4)
Learning rate (α)	$[10^{-5}, 10^{-3}]$	4.26×10^{-4}	3.04×10^{-4}	3.00×10^{-4}	4.23×10^{-5}	8.8×10^{-5}	5.25×10^{-5}
Entropy regularization (β_E)	$[10^{-5}, 10^{-10}]$	4.74×10^{-8}	1.43×10^{-7}	2.10×10^{-7}	2.77×10^{-9}	5.43×10^{-9}	4.19×10^{-10}
Length of the Episode buffer (m)	$[1, \cdots, 200]$	197	107	46	62	87	63
Action Space		$[0, 1, \cdots, 20]$	$[0, 1, \cdots, 20]$	$[0, 1, \cdots, 20]$	$[0, 1, \cdots, 20]$	$[0, 1, \cdots, 20]$	$[0, 1, \cdots, 20]$

Table 2 Hyperparameters and action space of the A3C algorithm in our six lost sales inventory model settings.

dynamic program can be solved. Then we can compare the optimality gaps of the A3C algorithm with the optimality gaps of state-of-the-art heuristic policies and approximate dynamic programming methods in Section 5.1. The second experiment considers larger settings where we can no longer solve for the optimal policy so we can only evaluate A3C relative to other heuristics in Section 5.2. We also provide sensitivity with respect to the demand (by testing different distributions including larger supports), lead time (and lead time uncertainty) and service levels.

5.1. Optimality gaps of the A3C algorithm and benchmarks for lost sales

We adopt six of the numerical experiments of Zipkin [2008a] to compare the A3C algorithm with several well-performing heuristics and methods using approximate dynamic programming and to stringently compare versus the optimal policy. All experiments have a unit holding cost $h = 1$, ordering cost $c = 0$ and demand is Poisson distributed with $\lambda = 5$. Lead times are 2, 3 and 4 periods and the shortage penalty p is either 4 or 9, resulting in six settings. The hyperparameters of the A3C algorithm are automatically tuned as described in Section 4. Table 2 reports the tuning ranges, the best performing hyperparameters and the action space we used in each experiment.

We benchmark the A3C algorithm against six heuristic policies. Three policies are simple to implement and possess interesting asymptotic behaviour: a Base-Stock policy, a Constant Order policy and a Capped Base-Stock policy. In addition we add three ADP policies: the Myopic 1-period and Myopic 2-period policies that have been shown by Zipkin [2008a] to perform well across a variety of settings, as well as the LP-ADP approach of Sun et al. [2016].

The optimal cost for each scenario was determined by numerically solving the DP using a discount factor of 99.9%. The costs obtained by the A3C algorithm are found by simulation and shown with 95% confidence intervals. The other policies' parameters are optimized by simulation, after which the steady state distribution of their Markov chain was computed to compute its expected cost.

Figure 2 plots the results of our numerical experiment. The A3C algorithm performs in line with the Myopic 1-period policy but cannot match the performance of the Myopic 2-period or the Capped Base-Stock policy. It does perform better than the Base-Stock and Constant Order policies. We can thus conclude that the A3C algorithm, with limited modification, develops good policies for the lost sales inventory problem, yet cannot beat the best performing heuristics.

Figure 3 reports the performance of 100 runs of the LP-ADP approach of Sun et al. [2016] with $p = 4$ and $l = 4$. One run corresponds to sampling 6,000 demands. Then we use the myopic policy

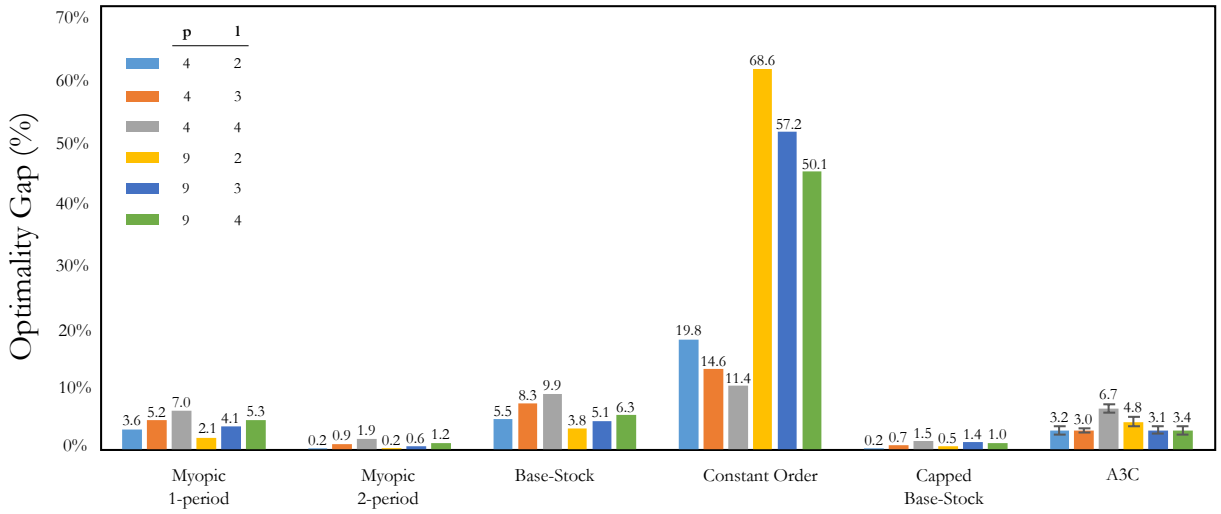

Figure 2 **The A3C algorithm performs in line with the Myopic 1-period policy but cannot match the performance of the Myopic 2-period policy nor of the even better performing Capped Base-Stock policy. Yet A3C does perform better than the Base-Stock and Constant Order policy.**

and store the 6,000 states visited and actions taken to construct the LP. We exclude the results of the first 1,000 periods warm-up. Then we solve the LP to obtain the coefficients of the quadratic approximation. Finally, we evaluate the performance of the LP-greedy policy for 60,000 periods, excluding a warm-up of 10,000 periods. This concludes one LP-ADP run.

It is striking in Figure 3 that the results are very run dependent: only a minority of the runs achieves reasonable performance yet no run outperformed the myopic sampling test policy. Sun et al. [2016] do report occasional improvements over the sampling test policy. While LP-ADP has been reported to obtain results in line or slightly better than the sampling test policy, in our experience it does not appear to be a stable, nor well-performing method for our lost sales inventory problem. We tested different lengths of sample paths but did not observe any improvement in performance. The benefit of LP-ADP is that it is computationally less intensive than A3C. One LP-ADP run can typically be finished in minutes instead of hours. Even 100 runs was easily done on one personal computer without requiring cloud computing.

The automatic tuning of the hyperparameter values revealed that the A3C performance was fairly insensitive to the length of the buffer size and the set of best performing values of the learning rate and entropy can be enveloped by an ellipsoid. Figure 4 shows the optimality gaps of each of the training runs of the A3C algorithm. We use a logarithmic scale as a significant amount of runs have very high optimality gaps. The red dots indicate policies that diverged to policies that reach infinitely high or low inventory levels such that the optimality gap is effectively infinitely large. In our visual their optimality gaps are around $10^4\%$ as we bound the state space for numerical reasons

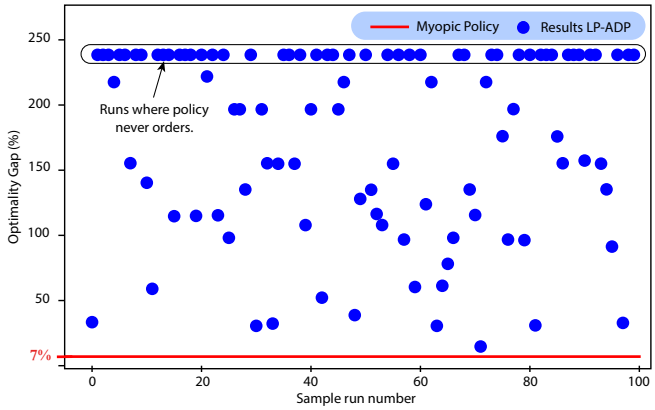

Figure 3 **Sample run performance of the LP-ADP method relative to the myopic base policy for a lost sales problem with $p = 4$ and $l = 4$.**

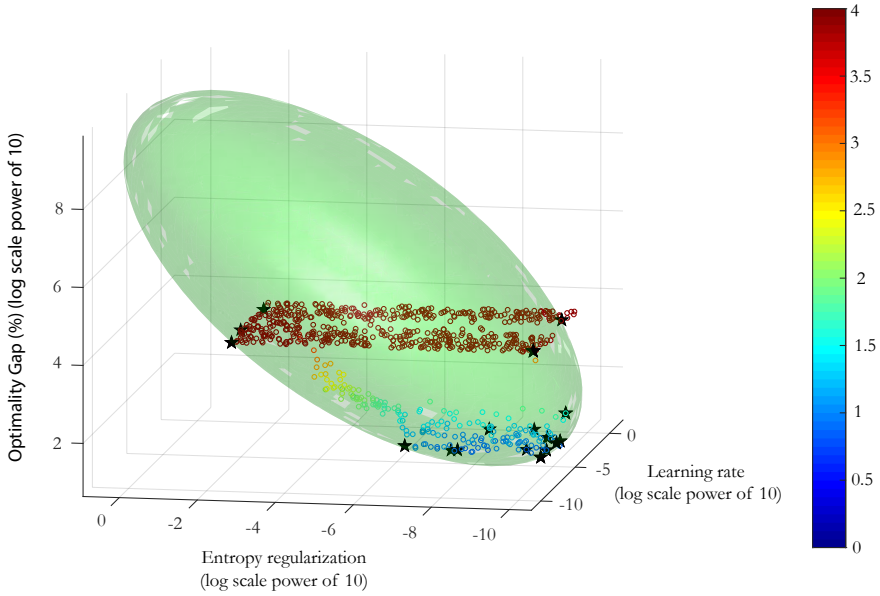

Figure 4 **The automatic tuning of the learning rate and the entropy over the intervals specified in Table 1 reveals that all points on the convex hull (marked with black star) fit an ellipsoid (results shown for $p = 4$ and $l = 4$). The red dots indicate policies where the A3C algorithm does not develop a good policy, e.g., policies that never place orders, or policies that result in infinite inventories.**

(i.e., we do not want to feed infinite values to the neural net). The best learning rate seems to be in the neighbourhood of 10^{-4} while the entropy factor should be set smaller than 10^{-7} to achieve good performance. We have used the same hyperparameter values in Section 5.2.

5.2. Sensitivity analysis

In what follows we provide numerical sensitivity of the performance of the A3C algorithm for larger demand support, longer (and stochastic) lead times and increasing service levels. Many of these settings no longer allow for a comparison with the optimal policy due to the curse of dimensionality. In fact, also the methods that rely on approximate dynamic programming (i.e., the myopic policies and the LP-ADP method) suffer in these problem settings. To facilitate comparison across our sensitivity analysis we thus only compare the A3C algorithm with the Constant Order, the Base-Stock and the Capped Base-Stock policy. Given its excellent performance in Figure 2, the latter serves as a solid benchmark while the other two are easy-to-use with interesting asymptotic properties. All reported costs are computed by simulating 100 sample paths of 100,000 periods and include 95% confidence intervals. The parameters of our base case (setting 3 in Figure 2) are: $h=1$, $c=0$, $l=4$, $p=4$ while demand is Poisson distributed with $\lambda=5$.

Sensitivity to longer lead times Figure 5 plots the results when the lead time is increased from 5 up to 10 periods. Similar to Figure 2, the Capped Base-Stock policy outperforms the A3C algorithm for small lead times. Interestingly, the A3C algorithm performs particularly well for large lead times, outperforming the benchmark policies. The reason for this excellent performance is related to the fact that we adopt a discrete demand distribution while the asymptotic properties of the Constant Order and Capped Base-Stock policies only hold for continuous demand supports. Due to the integer-spaced and small demand support, the Constant Order policy was unable to order close enough to the mean demand. (Our mean in the base case is 5; constantly ordering 5 units would result in large inventory holding costs while ordering 4 units causes on average 20% of the demand to be lost.) The A3C algorithm, however, converges to a policy that alternates between ordering four and five units such that its average order quantity is closer to the mean demand. This inspired us to design an additional mixed policy that orders either 4 or 5 units according to a Bernouilli distribution $\mathbb{P}[q_t=4]=P$ and $\mathbb{P}[q_t=5]=(1-P)$ and we numerically search over P. Figure 5 shows that this mixed strategy policy performs better than the benchmark policies and matches the performance of the A3C algorithm.

We acknowledge that these findings relate to our use of a small and integer demand support and that the discretization effect will fade away for larger demand supports. Yet, we find it interesting to show how the A3C algorithm may inspire the development of new benchmark heuristics.

Sensitivity to larger demand support To demonstrate how the A3C algorithm performs on larger problem settings we let the mean demand increase towards 25 in steps of five. To cope with the larger demand we enlarged the action space of the A3C algorithm to the range $[0, 100]$

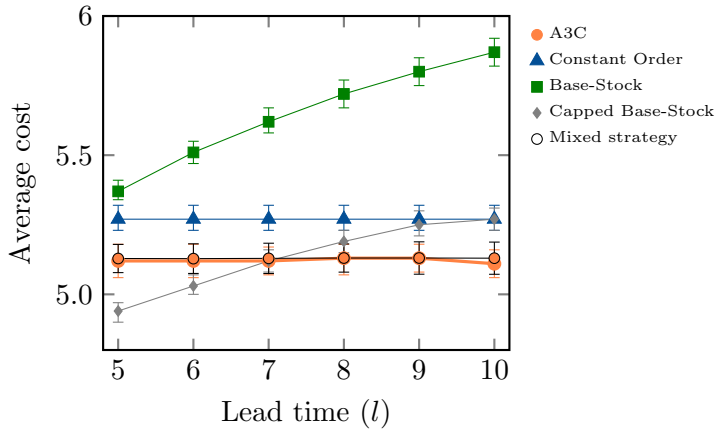

Figure 5 **Increasing the lead time favors the A3C algorithm, ultimately outperforming all benchmarks.**

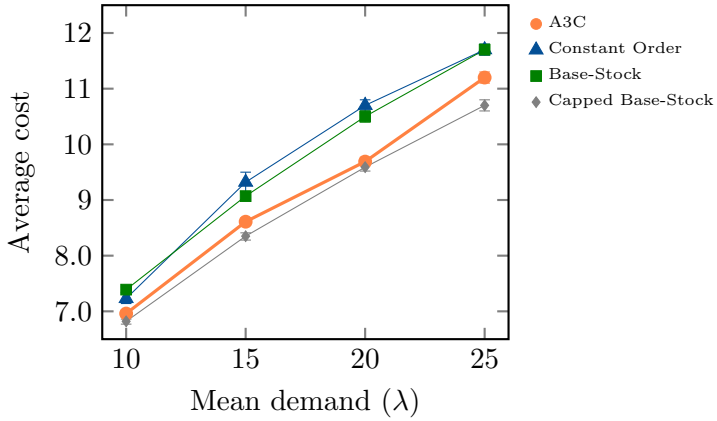

Figure 6 **Increasing the mean demand does not significantly impact the performance of the A3C algorithm.**

for this analysis. We observe in Figure 6 how a larger demand support does not significantly alter the relative performance compared to the benchmark policies. For this specific cost and lead time setting the A3C algorithm continues to perform slightly worse than the Capped Base-Stock policy but outperforms the constant-order and base-stock policy. This supports our findings in Section 5.1 and provides confidence that the A3C algorithm performs well in large settings.

We also tested the impact of different demand distributions, i.e., a uniform and a geometric distribution but this did not yield clear-cut insights. It depends on the benchmark and problem setting whether the relative performance of the A3C improves or reduces.

Sensitivity to different service levels We subsequently assess the impact of the service level by letting the lost sales penalty p vary between $\{4, 9, 19, 49\}$. The A3C algorithm performs in line with the Capped Base-Stock and the regular Base-Stock policy, which are both asymptotically optimal for large penalty costs. The A3C algorithm is able to develop policies close to these

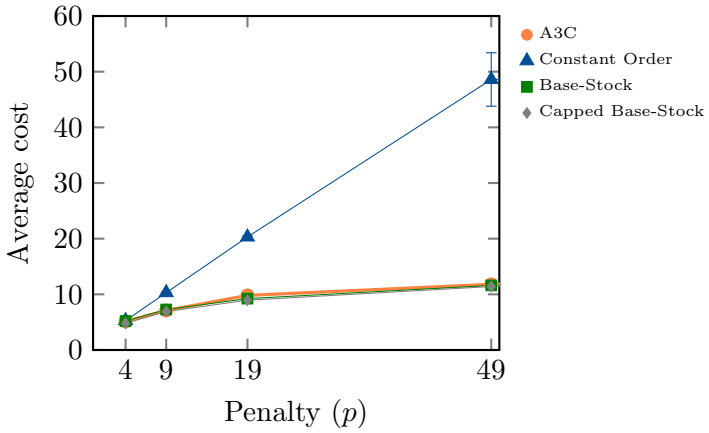

Figure 7 **Increasing the penalty cost (and thus service level) shows how the A3C algorithm performs in line with the (Capped) Base-Stock policy. The Constant Order policy suffers when the lost sales penalty rises.**

asymptotically optimal heuristics, performing for instance within 1% for the highest penalty cost we tested ($p = 99$).

Inclusion of stochastic lead times We also analyzed the performance under stochastic lead times. We uniformly sample the delivery lead time of each order from a discrete set of realizations on the integer space $[\underline{l}, \bar{l}]$, where the decision-maker does not have access to the arrival times of individual orders. At the beginning of period t the decision-maker thus observes the inventory before ordering (consisting of the ending inventory of last period plus all orders that arrive in period t) and the pipeline vector including all outstanding orders that have *not* arrived yet. In addition to learning the demand distribution, the model should thus also learn how to cope with uncertainty in the lead time.

We use the same base case and sample the lead times from a uniform distribution on the interval $[\underline{l}, \bar{l}]$. We fix $\underline{l} = 2$ and let $\bar{l}$ increase from 4 to 11. To the best of our knowledge no tailored heuristics exist in the lost sales setting with stochastic lead times. Janakiraman and Roundy [2004] investigate base-stock policies in this setting (albeit without order cross-overs) due to their simplicity — despite being sub-optimal. We thus compare against the same benchmark policies as the previous sensitivity experiments.

Our results demonstrate that the Capped Base-Stock policy continues to outperform even for stochastic lead times. Figure 8 demonstrates how the A3C algorithm performs in line with a constant-order policy, outperforms the Base-Stock policy and performs slightly worse than the Capped Base-Stock policy in all but one setting. The fact that the setting with $\bar{l} = 8$ does approach the Capped Base-Stock policy indicates there may be potential for further improvement in the other settings as well by further tuning the algorithm.

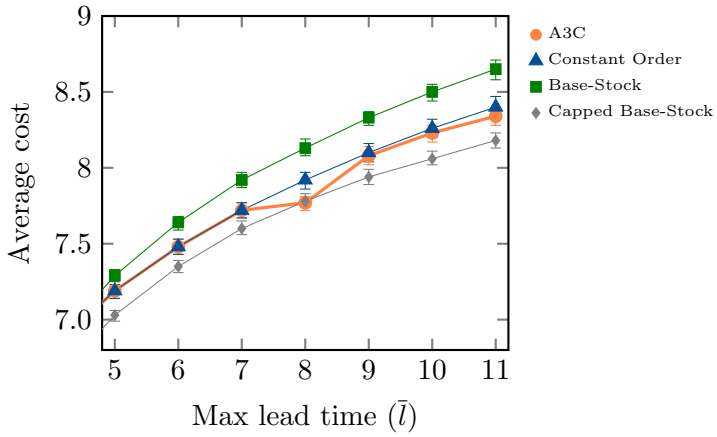

Figure 8 **When lead times are stochastic the A3C algorithm outperforms a simple Base-Stock policy, performs in line with a Constant Order policy but can not outperform a Capped Base-Stock policy in all evaluated settings. Additional tuning may reduce the gap with the Capped Base-Stock policy.**

6. Performance Evaluation of DRL in Dual Sourcing Inventory Models

Problem setting and Formulation In the conventional dual sourcing or dual mode model, inventory can be replenished at unit cost c_r using a regular source with lead time l_r *and* using an expedited source with lead time $l_e < l_r$ at premium unit cost $c_e > c_r$. At the beginning of any period t, two order quantities, $q_t^r \geq 0$ and $q_t^e \geq 0$, must be decided knowing the last observed inventory on hand, I_{t-1}, and outstanding receipts or "pipeline" vector $Q_{t-1} = (q_{t-l_r}^r + q_{t-l_e}^e, q_{t-l_r+1}^r + q_{t-l_e+1}^e, \cdots, q_{t-l_r+l_e-1}^r + q_{t-1}^e, q_{t-l_r+l_e-2}^r, \cdots, q_{t-1}^r)$. We note that this formulation assumes strictly positive lead times. If l_e=0, there are no outstanding expedited orders. After the order decision, orders $q_{t-l_r}^r$ and $q_{t-l_e}^e$ are received and added to the on-hand inventory. Finally, the unknown demand d_t is realized and subtracted from the on-hand inventory. Excess demand is backlogged so that the inventory and pipeline vector evolve as $I_t = I_{t-1} + q_{t-l_r}^r + q_{t-l_e}^e - d_t$ and $Q_t = (q_{t-l_r+1}^r + q_{t-l_e+1}^e, q_{t-l_r+2}^r + q_{t-l_e+2}^e, \cdots, q_{t-l_r+l_e}^r + q_t^e, q_{t-l_r+l_e-1}^r, \cdots, q_t^r)$.

The dual sourcing problem can be modeled as a Markov Decision Process with state $S_t = (I_{t-1}, Q_{t-1})$ at time t. Similarly to the lost sales system we can add orders placed l_r/l_e periods ago to the inventory on hand such that the state space becomes l_r-dimensional: $S_t = (s_0 = I_{t-1} + q_{t-l_r}^r + q_{t-l_e}^e, s_1 = q_{t-l_r+1}^r + q_{t-l_e+1}^e, s_2 = q_{t-l_r+2}^r + q_{t-l_e+2}^e, \cdots, s_{l_e} = q_{t-l_r+l_e}^r + q_{t-1}^e, s_{l_e+1} = q_{t-l_r+l_e-1}^r, \cdots, s_{l_r+1} = q_{t-1}^r)$. We note that future costs in periods t to $t+l_e-1$ cannot be influenced and are thus sunk at time t. Hence the state space dimension of the dynamic program can be reduced to $l_r - l_e$ [Sheopuri et al. 2010]. The action taken in period t is now two-dimensional: $a_t = (q_t^r, q_t^e)$ consisting of the ordered quantities from the regular and expedited source, respectively.

Results of Applying DRL We evaluate the performance of DRL in the six small-scale dual sourcing settings of Veeraraghavan and Scheller-Wolf [2008] with linear sourcing costs for which the

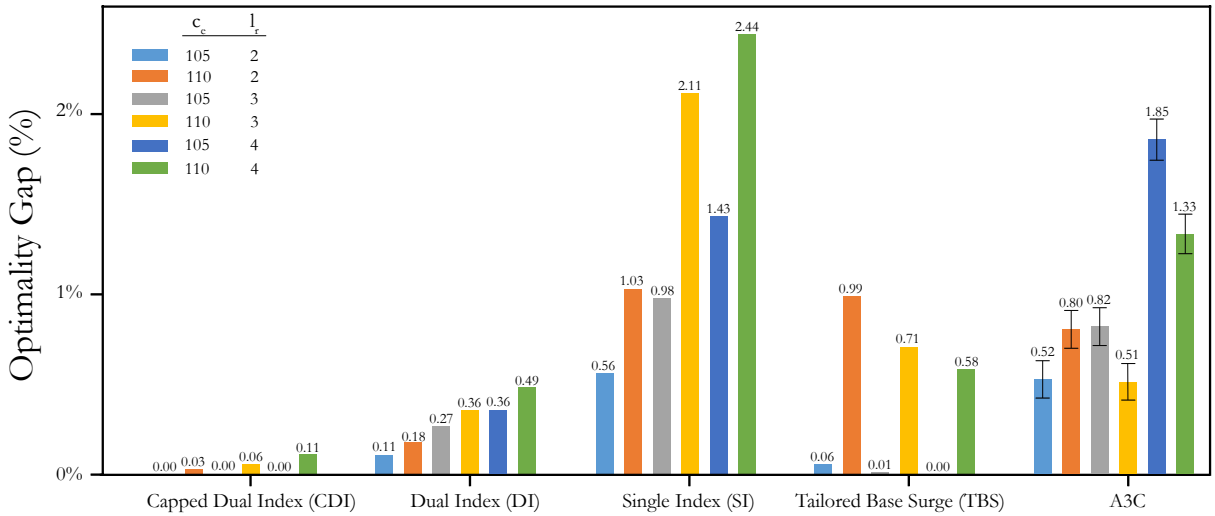

Figure 9 **The A3C algorithm is able to develop policies whose expected cost is within 2% of the optimal cost. This performance is comparable to Tailored Base-Surge, Single Index and Dual Index policies, but uniformly dominated by the Capped Dual Index policy (as expected given its robust optimality). Single and Dual Index performance deteriorates as the lead time increases while TBS improves.**

dynamic program is numerically solvable. The results are compared versus well-known heuristics. Since the value function of the dual sourcing problem is L♮-convex, we also benchmark against the LP-ADP technique, used by Chen and Yang [2019] in dual sourcing with supply uncertainty.

To keep the dynamic program computationally solvable, the lead time of the expedited source is 0 with a discrete uniform demand over $\{0, 1, 2, 3, 4\}$. Unit holding and backlog costs in all six experiments are $h = 5$ and $b = 495$ while the unit sourcing cost of the regular source $c_r = 100$. The six experiments range in their unit expediting sourcing cost c_e (high = 110 or low =105) and the regular source's lead time l_r (small = 2, medium = 3 and high = 4). All results are obtained as described in the lost sales problem in Section 5.1. Figure 9 reports our results and shows that the A3C algorithm succeeds in developing a policy that performs within 2% of optimality in all six experiments. This performance is comparable to the Single Index and Tailored Base Surge policies. The Capped Dual Index policy, which is robustly optimal, also outperforms all other policies in this stochastic setting. The optimality gaps are smaller compared to the lost sales settings we tested (as is also apparent by the scale of the vertical axis). This may be because all policies use a base-stock policy to control the local orders, which may reduce the cost differences between our benchmarks.

We also compare against the LP-ADP algorithm that Chen and Yang [2019] used in a dual sourcing setting with supply uncertainty. We used the SI, DI, CDI and TBS policies as sample test policies, and for each sample path we solved the LP and evaluated the LP-greedy policies. We repeated this procedure for 100 different sample paths and tested sample paths of lengths 6,000,

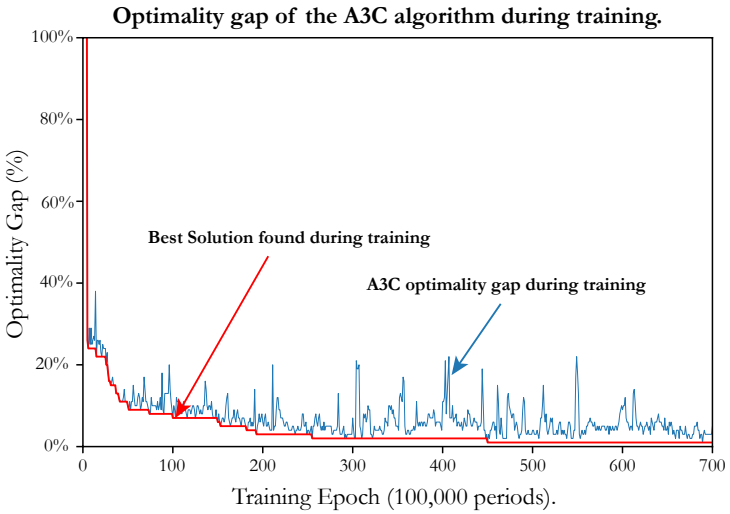

Figure 10 **The A3C algorithm quickly converges to a decent policy (within $\approx 20\%$ of optimality). Then it trains fast ($\approx$ 200 episodes of length 100,000 periods) to around 5%, after which it steadily reduces the optimality gap to less than 2% while exploring policies that fall within 10-20% of the optimal policy.**

11,000 and 60,000. Unfortunately, LP-ADP using this procedure did not manage to develop policies for dual sourcing that outperformed the test policies. Rather, the LP-ADP policies were often too myopic in nature; e.g., many policies did not order, or placed only expedited orders.

A potential reason may be that, in contrast to Chen and Yang [2019], our model contains no supply uncertainty and perhaps the latter is needed for LP-ADP to provide improvement over the test policies. Or perhaps the LP-ADP improvement documented by Chen and Yang [2019] in a setting with supply uncertainty resulted from using test policies that were not developed for a dual-sourcing setting with supply uncertainty. We conclude that, in our experience, the LP-ADP method was highly dependent on the sample test policy and failed to improve on the benchmark dual-sourcing heuristics that we used as sample test policies.

Noteworthy is that the training time of A3C remains relatively stable independent on the slow source's lead time, in contrast to dynamic programming. Nonetheless, it is not exceptional that more than five million periods are simulated per agent to find near-optimal policies, even in small-scale experiments. Figure 10 plots an example of a training curve of the A3C algorithm, where the best policy is obtained after 4.5 million periods of simulation per agent. Typically, the algorithm converges quickly to a relatively good policy (within $\approx 20\%$ of optimal cost). Then it trains fast ($\approx$ 200 episodes of length 100,000 periods) to approximately 5%, after which it steadily reduces the optimality gap to less than 2%. These results were obtained by occasionally restarting trained models with different hyperparameter values. For instance, a smaller learning rate could occasionally improve performance of these pre-trained models.

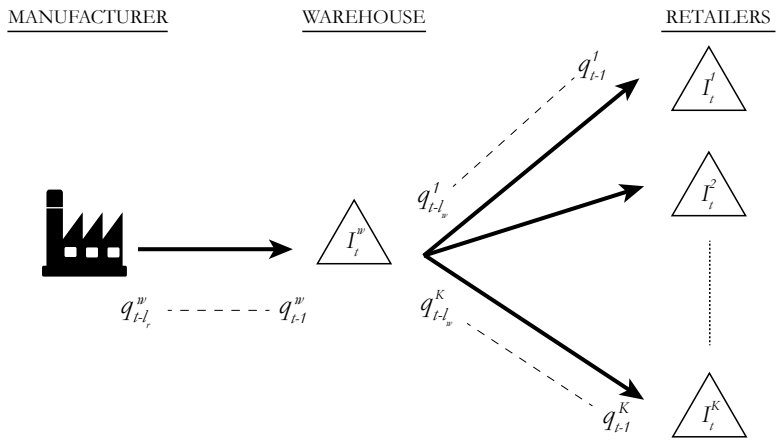

Figure 11 **Two-echelon supply chain of Van Roy et al. [1997] in which one capacitated warehouse is replenished from a manufacturer with lead time l_r and replenishes K capacitated retailers with lead time l_w.**

7. Performance Evaluation of DRL in Multi-echelon Inventory Models

Problem setting and Formulation We adopt the multi-echelon inventory model of Van Roy et al. [1997] with one warehouse and K identical retailers (see Figure 11). Van Roy et al. [1997] are the first to adopt a neuro-dynamic programming approach in inventory management. They use a neural net to approximate the value function which is updated using temporal difference learning. Employing the same setting allows us to show how a deeper neural net and a more advanced algorithm eliminates the need for manual feature engineering, which was needed to make the algorithm of Van Roy et al. [1997] work well.

Inventory held at the central warehouse or at the retailers incurs a unit holding cost h_w and h_r, respectively. The replenishment lead time from the manufacturer to the warehouse is l_w periods, and from the warehouse to any retailer l_r periods. All locations have limited capacity: (1) maximum production rate is C^m units per period; (2) warehouse inventory position cannot exceed C^w units; and (3) retail inventory position cannot exceed C^r units.

The model's MDP has state $S_t = (I^w_{t-1}, Q^w_{t-1}, I^r_{t-1}, Q^r_{t-1})$, where I^w_{t-1} is the end-of-period inventory at the warehouse and $Q^w_{t-1} = (q^w_{t-l_w}, \cdots, q^w_{t-1})$ its outstanding orders; $I^r_{t-1} = (I^1_{t-1}, \cdots, I^K_{t-1})$ the end-of-period inventories at the K retailers, and $Q^r_{t-1} = (q^1_{t-l_r}, \cdots, q^1_{t-1}, q^2_{t-l_r}, \cdots, q^2_{t-1}, \cdots, q^K_{t-l_r}, \cdots, q^K_{t-1})$ their respective outstanding orders.

The action vector $a_t = (q^w_t, q^1_t, \cdots, q^K_t)$, consists of the order quantities: q^w_t at the warehouse and q^i_t $(i = 1, \cdots, K)$ at retailer i. The retailer order quantities cannot exceed the end-of-period inventory on hand in the warehouse, i.e., $\sum_{i=1}^{K} q^i_t < I^w_{t-1} + q^w_{t-l_w}$. The sequence of events is as follows. Each period t demand d_i at retailer i is fulfilled by the inventory on hand at retailer i. In case of a stock-out, $[d_i - (I^i_{t-1} + q^i_{t-l_r})]^+ > 0$, customers either decide to wait for an expedited, same-day delivery from the warehouse — this occurs with probability P_w— or walk away such that

the retailer incurs a lost sale with probability $1-P_w$. The system is thus a hybrid of a backlogging and lost sales model. Important is that expedited deliveries are shipped *after* retailers have placed orders $(q_t^1,\cdots,q_t^K)$ and are allocated inventory from the warehouse. This implies that only $[I_{t-1}^w + q_{t-l_w}^w - \sum_{i=1}^{K} q_t^i]$ units can be expedited. Let B_t denote the number of customers in period t that request an expedited delivery. The number of requested expedited deliveries exceeding the available end-of-period inventory on hand at the warehouse, $\left[B_t - [I_{t-1}^w + q_{t-l_w}^w - \sum_{i=1}^{K} q_t^i]\right]^+$, is lost and the remainder, $B_t - \left[B_t - [I_{t-1}^w + q_{t-l_w}^w - \sum_{i=1}^{K} q_t^i]\right]^+$, is expedited and shipped same-day from the warehouse. Expedited deliveries by the warehouse incur a unit cost c_w while unmet demand is lost at a unit penalty cost p. The cost in period t thus is:

$$c_t(S_t,a_t) = h_w I_t^w + h_r \sum_{i=1}^{K} I_t^i + c_w \left[B_t - \left[B_t - [I_{t-1}^w + q_{t-l_w}^w - \sum_{i=1}^{K} q_t^i]\right]^+\right] + p\left[\left[B_t - [I_{t-1}^w + q_{t-l_w}^w - \sum_{i=1}^{K} q_t^i]\right]^+ + \sum_{i=1}^{K}\left[d_i - I_{t-1}^i + q_{t-l_r}^i\right]^+ - B_t\right], \quad (2)$$

where the first two elements represent the inventory holding costs in the supply chain, the third element the expedited delivery costs and the last two elements the lost sales costs upon stock-out at the warehouse and upon stock-out at the retailer. The state evolves as follows: $I_t^w = I_{t-1}^w + q_{t-l_w} - \sum_{i=1}^{K} q_t^i - \left[B_t - \left[B_t - [I_{t-1}^w + q_{t-l_w}^w - \sum_{i=1}^{K} q_t^i]\right]^+\right]$, $I_t^i = [I_{t-1}^i + q_{t-l_r}^i - d_i]^+$ for each retailer i, $Q_t^w = (q_{t-l_w+1}^w,\cdots,q_t^w)$ and $Q_t^i = (q_{t-l_r+1}^i,\cdots,q_t^i)$ for each retailer i.

Results of Applying DRL To apply A3C in the multi-echelon inventory model, we adapt the width of the first layer of the neural net to the dimension of the state space $(l_w + Kl_r)$. As the action space is K-dimensional, we reduce the action space by adopting a base-stock replenishment policy for the warehouse and the retailers with state-dependent base-stock levels, similar to Van Roy et al. [1997]. This reduces the action space to only two dimensions $a_t = [y_{S_t}^w, y_{S_t}^r]$, consisting of a base-stock level at the warehouse and a base-stock level for all retailers, which all depend on the state S_t. The warehouse order q_t^w raises its inventory position: $q_t^w = y_{s_t}^w - (I_{t-1}^w + \sum_{i=1}^{l_w} q_{t-i}^w)$ to its base-stock level $y_{s_t}^w$. Note that q_t^w may not exceed production capacity C^m and the inventory position may not exceed C^w. Hence,

$$q_t^w = \min\left(C^w - (I_{t-1}^w + \sum_{i=1}^{l_w} q_{t-i}^w), \min\left(y_{s_t}^w - (I_{t-1}^w + \sum_{i=1}^{l_w} q_{t-i}^w), C^m\right)\right).$$

After the warehouse has ordered, each retailer places its order to raise their inventory position to the base-stock level $y_{S_t}^r$: $q_t^i = y_{S_t}^r - (I_{t-1}^i + \sum_{i=1}^{l_r} q_{t-i}^i)$. After ordering, each retailer's inventory position may not exceed C_r:

$$q_t^i \le C^r - (I_{t-1}^i + \sum_{i=1}^{l_r} q_{t-i}^i).$$

Hyperparameters	**Tuning Range**	**Setting 1**	**Setting 2**
Learning rate used	$[10^{-5}, 10^{-3}]$	5.78×10^{-5}	1.74×10^{-4}
Entropy regularization (β_E)	$[10^{-10}, 10^{-6}]$	4.68×10^{-5}	1.46×10^{-8}
Length of the Episode buffer (m)	100	100	100
Base-stock levels warehouse ($y^w_{S_t}$)		$[50, 60, \cdots, 100]$	$[50, 60, \cdots, 100]$
Base-stock levels retailers ($y^r_{S_t}$)		$[0, 5, \cdots, 40]$	$[0, 5, \cdots, 50]$

Table 3 Hyperparameters of the A3C algorithm to achieve results in both multi-echelon settings.

	l_w	l_r	μ	σ	K	h_w	h_r	c_w	p	P_w	C_p	C_w	C_r
Setting 1	2	2	5	14	10	3	3	0	60	0.8	100	1000	100
Setting 2	5	3	0	20	10	3	3	0	60	0.8	100	1000	100

Table 4 Numerical setting used in lost sales multi-echelon setting of Van Roy et al. [1997]

If the warehouse has insufficient capacity to deliver all retailer orders, the inventory is allocated to the retailers by minimizing the difference in inventory position between all retailers after orders are allocated following Van Roy et al. [1997].

We test the two numerical settings used in Van Roy et al. [1997] (see Table 4) Each element of the random demand vector $D = (d_1, \cdots, d_K)$ is sampled from a normal distribution with mean μ and standard deviation σ, rounded to the nearest integer. As computing the optimal policy is intractable we only compare the A3C versus a base-stock policy with constant, i.e., not state-dependent, base-stock levels y_w and y_r. Unlike the dual sourcing and lost sales inventory models we cannot deploy the LP-ADP approach, as we do not know the structure of the value function, and the K-dimensional demand makes the number of transition probabilities too large and the formulation of the constraints of the LP infeasible.

We trained the A3C algorithm using the automatic tuning process with the same set of hyper-parameters as in dual sourcing and lost sales (see Table 3). We fixed the buffer size at 100, which provided decent results after 100 training runs of the A3C algorithm. The discrete action space of the A3C algorithm is identical to that used in Van Roy et al. [1997]. The state-dependent base-stock levels $y^w_{S_t}$ are restricted to 6 values $\{50, 60, \cdots, 100\}$ while $y^r_{S_t}$ is restricted to 9 values $\{0, 5, 10, \cdots, 40\}$ in Setting 1 and 11 values $\{0, 5, 10, \cdots, 50\}$ in Setting 2. Restricting the base-stock levels to this set of values will go at the expense of being able to reach the optimal policy. Yet, it allowed us to improve on a state-independent base-stock policy in which all base-stock levels are allowed. We find that the A3C algorithm improves approximately 9% and 12% on the base-stock policy with constant base-stock levels. These results are in line with, and slightly better in setting 2, than Van Roy et al. [1997] who report savings around 10%.

One key distinguishing factor with the method of Van Roy et al. [1997] is that A3C (or DRL) does not require manual feature engineering. Van Roy et al. [1997] manually develop 23 features based on the state vector. Instead, DRL or A3C learns directly from the state vector. This is

possible because our network uses four layers and is 'deeper' than the single layer with activation neural net used in Van Roy et al. [1997]. Their careful selection of the features has resulted in excellent performance but required problem knowledge. In contrast, DRL does not require any manual feature engineering, a desirable characteristic of a general purpose technology. Moreover, we observe that A3C manages to find slightly better policies in the larger setting 2.

8. Conclusions and Reflections

Our study provides evidence that deep reinforcement learning can effectively solve classic, intractable inventory problems. The A3C algorithm can be trained to produce policies that match the performance of state-of-the-art heuristics and other approximate dynamic programming methods. After extensive manual tuning of 9 hyperparameters in one problem setting, we found that fixing 6 parameters and automatically tuning the learning rate, entropy regularization, and buffer size resulted in good performance in the other two problem settings. With such minimal changes among three different problem settings, DRL thus seems a promising general purpose technology.

Yet we also found that A3C does not outperform all heuristic policies and remains, like other ADP methods, a black box. Tuning the hyperparameters of the neural network is effort-intensive and requires both art and science. The computational requirements are formidable: we had to resort to cloud computing.

So our conclusion is nuanced: DRL seems a promising general purpose technology that can be applied to intractable inventory problems. Yet, there remains a job for academic inventory researchers to generate structural policy insight or design specialized policies that are (ideally provably) near optimal.

Our study has focused on solving classic, stationary models of inventory management. Before applying DRL in practice, we should evaluate performance in non-stationary environments with multiple products and unknown demands that must be learned from historical data. We hope that by sharing our experience and code, this paper will provide a stepping stone for such research in inventory management and other operational settings.

References

Allon G, Van Mieghem JA (2010) Global Dual Sourcing: Tailored Base-Surge Allocation to Near- and Offshore Production. *Mgt Sci* 56(1):110–124.

Arrow KJ, Karlin S (1958) *Studies in the mathematical theory of inventory and production* (Stanford University).

Bellman R (1954) The Theory of Dynamic Programming. *Bulletin of the Amer Math Soc* 60(6):503–515.

Bergstra J, Bengio Y (2012) Random Search for Hyper-Parameter Optimization. *J. of Machine Learn. Res.* 13:281–305.

Bergstra J, Komer B, Eliasmith C, Yamins D, Cox DD (2015) Hyperopt: a python library for model selection and hyperparameter optimization. *Comput. Sci. & Discovery* 8(1):1749–4699.

Bergstra J, Yamins D, Cox DD (2013) Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, 115–123.

Brown DB, Smith JE (2014) Information relaxations, duality, and convex stochastic dynamic programs. *Ops Res* 62(6):1394–1415.

Chen W, Dawande M, Janakiraman G (2014) Fixed-dimensional stochastic dynamic programs: An approximation scheme and an inventory application. *Ops Res* 62(1):81–103.

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

Clark AJ, Scarf H (1960) Optimal Policies for a Multi-echelon Inventory Problem. *Mgt Sci* 6(4):363–505.

de Kok T, Grob C, Laumanns M, Minner S, Rambau J, Schade K (2018) A typology and literature review on stochastic multi-echelon inventory models. *Eur J of Op Res* 269(3):955–983.

Fang J, Zhao L, Fransoo JC, Van Woensel T (2013) Sourcing strategies in supply risk management: An approximate dynamic programming approach. *Comput. Oper. Res.* 40(5):1371–1382.

Federgruen A, Zipkin P (1984) Approximations of dynamic, multilocation production and inventory problems. *Mgt Sci* 30(1):69–84.

Fukuda Y (1964) Optimal Policies for the Inventory Problem with Negotiable Leadtime. *Mgt Sci* 10(4):690–708.

Giannoccaro I, Pontrandolfo P (2002) Inventory management in supply chains: a reinforcement learning approach. *Int. Journ. Prod. Econ.* 78(2):153–161.

Goldberg DA, Katz-Rogozhnikov DA, Lu Y, Sharma M, Squillante MS (2016) Asymptotic Optimality of Constant-Order Policies for Lost Sales Inventory Models with Large Lead Times. *Math. Oper. Res.* 41(3):898–913.

Halman N, Klabjan D, Mostagir M, Orlin J, Simchi-Levi D (2009) A fully polynomial-time approximation scheme for single-item stochastic inventory control with discrete demand. *Math. Oper. Res.* 34(3):674–685.

Hessel M, Modayil J, van Hasselt H, Schaul T, Ostrovski G, Dabney W, Horgan D, Piot B, Azar M, Silver D (2017) Rainbow: Combining improvements in deep reinforcement learning. *arXiv:1710.02298* .

Hua Z, Yu Y, Zhang W, Xu X (2015) Structural properties of the optimal policy for dual-sourcing systems with general lead times. *IIE Transactions* 47(8):841–850.

Huh WT, Janakiraman G, Muckstadt JA, Rusmevichientong P (2009) Optimality of Order-Up-To Policies in Lost Sales Inventory Systems. *Mgt Sci* 55(3):404–420.

Janakiraman G, Roundy RO (2004) Lost-sales problems with stochastic lead times: Convexity results for basestock policies. *Ops Res* 52(5):795–803.

Keller PW, Mannor S, Precup D (2006) Automatic basis function construction for approximate dynamic programming and reinforcement learning. *Proceedings of the 23rd International Conference on Machine Learning*, 449–456, ICML '06 (ACM).

Kingma DP, Ba J (2015) Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* .

Kohl N, Stone P (2004) Policy gradient reinforcement learning for fast quadrupedal locomotion. *Proceedings of the IEEE International Conference on Robotics and Automation.*

Levi R, Perakis G, Uichanco J (2015) The data-driven newsvendor problem: New bounds and insights. *Ops Res* 63(6):1294–1306.

Mnih V, Badia AP, Mirza M, Graves A, Lillicrap TP, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783* .

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. *Nature* 518:529–533.

Morton TE (1969) Bounds on the Solution of the Lagged Optimal Inventory Equation with no Demand-Backlogging and Proportional Costs. *SIAM Rev.* 11(4):572–596.

Morton TE (1971) The Near-Myopic Nature of the Lagged-Proportional-Cost Inventory Problem with Lost-Sales. *Ops Res* 19(7):1708–1716.

Murota K (2003) *Discrete Convex Analysis: Monographs on Discrete Mathematics and Applications 10* (Society for Industrial and Applied Mathematics).

Nahmias S, Smith SA (1993) *Perspectives in Operations Management* (Springer, Boston, MA).

Nahmias S, Smith SA (1994) Optimizing Inventory Levels in a Two Echelon Retailer System with Partial Lost Sales. *Ops Res* 40(5):582–596.

Oroojlooyjadid A, Nazari M, Snyder L, Takáč M (2017) A Deep Q-Network for the Beer Game with Partial Information. *arXiv:1708.05924* .

Powell WB (2011) *Approximate dynamic programming: solving the curses of dimensionality* (Wiley).

Puterman ML (1994) *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (Wiley).

Reiman MI (2004) A new and simple policy for the continuous review lost sales inventory model. *Working Paper, Bell Labs, Lucent Technologies* .

Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. *arXiv:1511.05952* .

Scheller-Wolf A, Veeraraghavan S, van Houtum GJ (2003) Effective dual sourcing with a single index policy. *Working Paper, Wharton at the University of Pennsylvania* .

Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P (2017a) Trust region policy optimization. *arXiv:1502.05477* .

Schulman J, Moritz P, Levine S, Jordan M, Abbeel P (2015) High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438* .

Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017b) Proximal policy optimization algorithms. *arXiv:1707.06347* .

Sheopuri A, Janakiraman G, Seshadri S (2010) New Policies for the Stochastic Inventory Control Problem with Two Supply Sources. *Ops Res* 58(3):734–745.

Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D (2016) Mastering the game of go with deep neural networks and tree search. *Nature* 529:484–489.

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, et al. (2017) Mastering the game of go without human knowledge. *Nature* 550:354.

Strehl AL, Li L, Wiewiora E, Langford J, Littman ML (2006) Pac model-free reinforcement learning. *Proceedings of the 23rd international conference on Machine learning*, 881–888.

Sun J, Van Mieghem JA (2019) Robust Dual Sourcing Inventory Management: Optimality of Capped Dual Index Policies. *Manufacturing Service Oper. Management* 21(4):713–948.

Sun P, Wang K, Zipkin P (2016) Quadratic Approximation of Cost Functions in Lost Sales and Perishable Inventory Control Problems. *Working Paper,Fuqua School of Business, Duke University* .

Sutton RS, Barto AG (1998) *Introduction to reinforcement learning*, volume 135 (MIT press Cambridge).

Tesauro G (1995) Temporal difference learning and td-gammon. *Communications of the ACM* 38(3):58–68.

van Hasselt H, Guez A, Silver D (2015) Deep reinforcement learning with double q-learning. *arXiv:1509.06461* .

Van Roy B, Bertsekas DP, Lee Y, Tsitsikis JN (1997) A Neuro-Dynamic Programming Approach to Retailer Inventory Management. *Proceedings of the IEEE Conference on Decision and Control* .

Veeraraghavan S, Scheller-Wolf A (2008) Now or Later: A Simple Policy for Effective Dual Sourcing in Capacitated Systems. *Ops Res* 56(4):850–864.

Wang Z, Schaul T, Hessel M, van Hasselt H, Lanctot M, de Freitas N (2015) Dueling network architectures for deep reinforcement learning. *arXiv:1511.06581* .

Watkins C (1989) *Learning from delayed rewards*. Ph.D. thesis, Cambridge University.

Whittemore AS, Saunders SC (1977) Optimal Inventory Under Stochastic Demand With Two Supply Options. *J on Appl Math* 32(2):293–305.

Williams R (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8(3):229–256.

Xin L (2019) Understanding the performance of capped base-stock policies in lost-sales inventory models. *Forthcoming in Operations Research. Available at SSRN: https://ssrn.com/abstract=3357241* .

Xin L, Goldberg DA (2017) Asymptotic Optimality of Tailored Base-Surge Policies in Dual-Sourcing Inventory Systems. *Mgt Sci* 64(1):437–452.

Zipkin P (2008a) Old and new methods for lost-sales inventory systems. *Ops Res* 56(5):1256–1263.

Zipkin P (2008b) On the Structure of Lost-Sales Inventory Models. *Ops Res* 56(4):937–934.