European Journal of Operational Research 298 (2022) 401–412

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Invited Review

Deep reinforcement learning for inventory control: A roadmap

Robert N. Boute[a,b,*], Joren Gijsbrechts[c], Willem van Jaarsveld[d], Nathalie Vanvuchelen[a]

[a] Faculty of Economics and Business, KU Leuven, Belgium
[b] Vlerick Business School, Belgium
[c] Universidade Católica Portuguesa, Portugal
[d] Eindhoven University of Technology, the Netherlands

ARTICLE INFO

Article history:
Received 16 May 2020
Accepted 6 July 2021
Available online 16 July 2021

Keywords:
Inventory management
Machine learning
Reinforcement learning
Neural networks

ABSTRACT

Deep reinforcement learning (DRL) has shown great potential for sequential decision-making, including early developments in inventory control. Yet, the abundance of choices that come with designing a DRL algorithm, combined with the intense computational effort to tune and evaluate each choice, may hamper their application in practice. This paper describes the key design choices of DRL algorithms to facilitate their implementation in inventory control. We also shed light on possible future research avenues that may elevate the current state-of-the-art of DRL applications for inventory control and broaden their scope by leveraging and improving on the structural policy insights within inventory research. Our discussion and roadmap may also spur future research in other domains within operations management.

1. Introduction

Reinforcement learning (RL) is an area of machine learning that focuses on sequential decision-making. It determines how to take actions in an environment to maximize some notion of cumulative reward. RL algorithms *prescribe* what to do; this differs from supervised and unsupervised learning, two other well-known areas of machine learning, that focus on description or prediction. The introduction of multi-layer perceptrons (MLPs), nowadays also referred to as deep neural networks, in supervised machine learning, has led to substantial performance improvements. A notable breakthrough was the impressive win in 2012 of a student team from the University of Toronto using neural networks in the visual object recognition competition ImageNet (Krizhevsky, Sutskever, & Hinton, 2012), such that the following year all top finalists used the then-novel deep learning approach (Gershgorn, 2018). Neural networks are now well-established in (deep) supervised learning, yet their inclusion in RL, known as deep reinforcement learning (DRL), remains notably harder. Garychl (2018) provides some successful applications of DRL in different fields, such as robotics, gaming, and traffic light control, despite several obstacles in the practical application of DRL (Dulac-Arnold, Mankowitz, & Hester, 2019).

One such application, at the heart of operations management, is inventory control, where replenishment actions must be taken to minimize costs. Despite decades of research, the optimal policy to many inventory control problems remains unknown. Their analytic intractability simply renders it impossible to derive closed-form expressions of the optimal policy structure and related parameters. Instead, numerical approaches, e.g., dynamic programming methods such as value or policy iteration (Puterman, 1994), may be used to solve small (discretized) settings to optimality. As the dimensions of the problem grow, however, these techniques also become numerically intractable, even for moderate-size problems. DRL can be used to develop near-optimal policies that are hard, if not impossible, to be obtained using conventional approaches, especially when little is known about the focal problem.

This paper provides a roadmap on how inventory control may benefit from DRL. The two main constituents of DRL are Markov Decision Processes (MDPs) and neural networks: MDPs are the model/language in which the *problems* solved with DRL must be expressed; neural networks are used both to express and to find approximate *solutions* to those problems. Section 2 introduces both. In Section 3 we provide an overview of the different design choices to set up a DRL algorithm for inventory control. Although DRL algorithms are publicly available off-the-shelf, the significant effort required to choose the correct design may prevent their widespread application in practice. We focus on those aspects of DRL which we believe to be most relevant for inventory control

* Corresponding author at: Vlerick Business School, Vlamingenstraat 83, BE-3000 Leuven, Belgium.
E-mail address: robert.boute@vlerick.com (R.N. Boute).

https://doi.org/10.1016/j.ejor.2021.07.016

and where we see most potential for future research. These future research avenues are further explored in Section 4.

2. Markov decision problems and neural networks

MDPs are widely used in many research domains, and particularly deep-rooted in the operations community. We give a brief introduction here, mainly to introduce notation and terminology that will be used throughout the paper; for in-depth discussions on MDPs we refer to Bertsekas (1987) and Puterman (1994).

2.1. Markov decision processes to optimize inventory control

At each time step t the MDP can be described by its state $s \in \mathcal{S}$, with $\mathcal{S}$ the set of states. An action a is then chosen from a set of feasible actions $\mathcal{A}$. A deterministic policy maps states to actions, such that it prescribes which action to take in each state: $\pi : \mathcal{S} \to \mathcal{A}$. A *stochastic* policy[1] prescribes the probability of each action in each state: $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$, where $\pi(a|s)$ is the probability of taking action a in state s. That is, if $a \sim \pi(\cdot|s)$ then a is the random action taken by π in s. When taking action a in state s, the probability to transition to state s' is denoted $\mathbb{P}(s' \mid s, a)$. Denote $c(s, a)$ the expected cost when taking action a from state s.

The expectation of the total discounted cost over an infinite horizon, when starting in state s and thereafter following policy π is captured by the *state-value function* $v^\pi(s)$. This value function of state s can be recursively related to the value functions of the states s' that may be reached from state s:

$$v^\pi(s) = \sum_a \pi(a|s) \left(c(s, a) + \gamma \sum_{s'} \mathbb{P}(s' \mid s, a) v^\pi(s') \right), \quad \forall s \in \mathcal{S}, \tag{1}$$

with $0 < \gamma \leq 1$ the discount factor. We tacitly assume a discrete action and transition space in our notation but note that replacing the summation with integrals provides us the continuous counterpart. In Section 3.5, we discuss the use of continuous action spaces. The cost-minimizing state-value functions $v^*(s)$ can readily be obtained by solving the famed *Bellman optimality equations* (Bellman, 1954):

$$v^*(s) = \min_a \left\{ c(s, a) + \gamma \sum_{s'} \mathbb{P}(s' \mid s, a) v^*(s') \right\}, \quad \forall s \in \mathcal{S}. \tag{2}$$

Related to the value functions, the *action-value functions* $q^\pi(s, a)$ define the future expected costs of taking action a in state s and thereafter reverting to policy π:

$$q^\pi(s, a) = c(s, a) + \gamma \sum_{s'} \mathbb{P}(s' \mid s, a) \sum_{a'} \pi(a'|s') q^\pi(s', a'), \quad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}. \tag{3}$$

The value and action-value functions only differ in the first action. In a similar way to the optimal state-values, the optimal action-values $q^*(s, a)$ satisfy: $q^*(s, a) = c(s, a) + \gamma \sum_{s'} \mathbb{P}(s' \mid s, a) v^*(s')$, $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.

The optimal policy π^* directly follows from the optimal value functions $v^*(s)$ or action values $q^*(s, a)$:

$$\pi^*(a \mid s) > 0 \to a \in \operatorname*{argmin}_a \left\{ c(s, a) + \gamma \sum_{s'} \mathbb{P}(s' \mid s, a) v^*(s') \right\}, \quad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A},$$

$$\to a \in \operatorname*{argmin}_a q^*(s, a), \quad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}. \tag{4}$$

In many problems, only one action will be cost-minimal leading to a deterministic policy. In rare cases, actions might be equivalently cost-efficient (i.e., the argmin of $q^*(s, a)$ contains multiple actions) such that the optimal policy may be stochastic. DRL algorithms will often develop stochastic policies as this may encourage exploration of new actions and facilitates gradient-based training. We further discuss this in Section 3.3 (Exploration versus exploitation).

Depending on the problem complexity, the optimal policy can sometimes be analytically characterized. For other cases, approximative analysis and/or heuristic arguments are used to identify a class of policies, and analytic approaches may be deployed to find good policy parameters within that class. Apart from these exact and approximative analytic approaches, MDPs can also be solved numerically exact or approximate. This yields four partially overlapping approaches to analyze and optimize inventory problems using MDPs:

1. **Analytical and Exact:** For relatively simple problems, the optimality Eq. (2) serve as a starting point for analytical characterization of optimal policies. In those cases the structure of the optimal policy is deeply entangled with the structure of the value function. For periodic-review inventory models with positive lead times and full backordering, for instance, the seminal work of Karlin & Scarf (1958) proves that the value function is convex such that the optimal policy structure satisfies a base-stock policy. Apart from giving insight into the optimal policy structure, these analytical results may also facilitate characterization of the optimal policy parameters: the optimal base-stock levels for full backordering systems can be expressed in closed form using newsvendor fractiles under certain conditions (see for instance Veinott, 1966).
2. **Analytical and Approximate:** For some MDPs, the optimal policies have a complex form that is analytically intractable. To derive some analytic insights, it is common to impose a certain class of policies that approximate the optimal policy. E.g., for assemble-to-order inventory systems it is common to assume base-stock policies even though the optimal policy is not a base-stock policy, yet elegant insights have been derived under this assumption (e.g. Lu & Song, 2005). Similarly, some classes of policies can be shown to be asymptotically optimal, such as constant order policies for the intractable lost sales problem with long lead times (Goldberg, Katz-Rogozhnikov, Lu, Sharma, & Squillante, 2016).
3. **Numerical and Exact:** Clean analytic characterizations of optimal solutions to (2) can only be obtained for specific problems under specific assumptions. When relaxing assumptions, these characterizations are at best *partial*, and finding optimal policies from (2) involves the use of general-purpose exact numerical methods such as policy iteration. The optimal policy for inventory models with lost sales and arbitrary lead times, for instance, does not have a simple form, but it can be obtained numerically for small problems with short lead times and limited demand support. The optimal policies then serve as a useful benchmark for heuristic policies. In case they can also be directly applied to business problems, they can be very useful, regardless of what is known about their structure (see e.g. Martagan, Krishnamurthy, Leland, & Maravelias, 2018). When these exact numerical techniques are not scalable to larger problem instances, as is the case for many inventory control problems, we have to resort to approximate methods.
4. **Numerical and Approximate:** Most practical problems formulated as MDPs *cannot* be solved using exact methods because they have too many states, actions or transitions. For example, the state space for lost sales inventory systems grows ex-

[1] For many MDPs, deterministic policies are optimal. We will use the more general notation of the stochastic policy, as they prove to be a crucial element in policy gradient methods (which we discuss later in more detail). Note that if a single action a is proposed per state and $\pi(a|s) = 1$, we obtain a deterministic policy.

ponentially in the lead time: application of exact methods becomes cumbersome for lead times above five and hopeless for lead times above ten. To analyse such problems, the Bellman optimality equations in (2) have been used as the basis for a vast range of numerical approaches that seek policies with satisfactory, yet sub-optimal performance. For such numerical approaches, the MDP framework is thus used as a *modelling* tool.

DRL belongs to the class of *approximate numerical methods*. Such methods have been referred to using various nomenclature such as approximate dynamic programming, neuro-dynamic programming and reinforcement learning, and these terminologies are essentially equivalent (cf. Bertsekas, 2019); in what follows we will adopt the term reinforcement learning. *Deep* reinforcement learning refers to the use of neural networks in reinforcement learning to approximate the value functions, the (stochastic) policy, or a combination of both.

The application of neural networks in MDPs is not new (see e.g., Bertsekas & Tsitsiklis, 1996; Powell, 2007; Tesauro, 1992; 1994), but the advances in hardware and software support for neural networks have invigorated their use: the highly parallel structure of modern GPUs has reduced the overall time needed to train neural networks by orders of magnitude in the last decade. In concert, deep learning frameworks such as PyTorch and Tensorflow have significantly simplified the implementation of efficient training algorithms that use such accelerators, resulting in a very broad research base and substantial algorithmic advances. A key breakthrough of the use of neural networks in decision-making was AlphaZero (Silver et al., 2017), which combines neural networks with clever search techniques to arrive at board game policies for various games that vastly outperform the state-of-the-art, without using any expert knowledge.

As we have recently witnessed some applications of DRL in inventory management, we believe the technique also bears substantial further promise for our field. The most apparent strength of DRL lies within its use of deep neural networks to model a diversity of (complex) problems without requiring extensive domain knowledge or making restrictive assumptions. This contrasts to classical inventory control solution methods that typically require thorough domain knowledge or rely on restrictive modelling assumptions. DRL may also provide new opportunities to enable more interaction between the numerical and analytical approaches described above. Simpler (explainable) heuristics may for instance be extracted from policies developed by DRL algorithms or (existing or new) heuristics may be benchmarked against DRL policies to build better understanding of their dynamics and performance (i.e., why or when do they perform better or worse). We also perceive DRL to become a complementary tool in data-driven inventory control as it enables turning data into decision-making. Think for instance of enriching the state space by big data or learning inventory policies based on previously collected datasets. We acknowledge, however, that DRL also has some disadvantages, such as: the learning process of DRL algorithms may not always be stable or convergent and its performance may be very sensitive to hyperparameters. Further research is thus needed to improve the scalability and training performance of DRL algorithms. Nonetheless, we believe DRL to be a promising avenue that can be used on a variety of decision-making problems (among which inventory control problems) while simultaneously blurring the lines between various solution approaches and paving the way for data-driven inventory control.

As the typical OR/OM researcher may not be familiar with neural networks, we continue with a brief introduction. In Section 3 we elaborate on their role in DRL algorithms for application in inventory control.

2.2. The use of neural networks to solve MDPs

Neural networks are essentially parameterized functions from $\mathbb{R}^N$ to $\mathbb{R}^M$ (with $N, M \in \mathbb{N}$). Denote such a (generic) function by $\mathcal{N}_\theta(\cdot)$, with θ the parameters of the function/network. The prototypical example of a neural network is the *multi-layer perceptron* (MLP). The MLP, like most neural networks, is composed of multiple operations, denoted as *layers*. Layer/operation $l \in \{1, \dots, L\}$ is parameterized by a (weight) matrix $A^{[l]}$ and a (bias) vector[2] $\mathbf{b}^{[l]}$, thus the parameters of the entire network are $\theta = \{A^{[1]}, \mathbf{b}^{[1]}, \dots, A^{[L]}, \mathbf{b}^{[L]}\}$. Each layer also involves an activation function $f^{[l]}(\cdot) : \mathbb{R}^k \to \mathbb{R}^k$; a typical example is $f^{[l]}(\mathbf{x}) = \max(\mathbf{x}, 0)$, where the max is taken component-wise. The output of each layer l of the MLP is obtained by operating on the output $\boldsymbol{\psi}^{[l-1]}$ of the previous layer, i.e., for $l \in \{1, \dots, L\}$[3]:

$$\boldsymbol{\psi}^{[l]} = f^{[l]}(A^{[l]}\boldsymbol{\psi}^{[l-1]} + \mathbf{b}^{[l]}).$$

The MLP is then defined by $\mathcal{N}_\theta(\mathbf{x}) := \boldsymbol{\psi}^{[L]}$, with $\boldsymbol{\psi}^{[0]} := \mathbf{x}$, such that $\boldsymbol{\psi}^{[1]} = f^{[1]}(A^{[1]}\mathbf{x} + \mathbf{b}^{[1]})$. In general, the MLP maps vectors $\mathbf{x} \in \mathbb{R}^N$ to vectors $\mathcal{N}_\theta(\mathbf{x}) \in \mathbb{R}^M$. For the final layer L, the activation function may be used to ensure that the output of the neural network satisfies certain constraints. E.g., when the output of the MDP must represent a probability distribution over $\{1, \dots, M\}$, outputs must lie on the standard/probability simplex Δ^M. This can be accomplished by setting the so-called *softargmax*/softmax function σ as activation function[4] for layer L, i.e., $f^{[L]}(\cdot) = \sigma(\cdot)$. Alternatively one can use no activation function for the final layer, which is equivalent to setting $f^{[L]}(\mathbf{x}) = \mathbf{x}$ and results in arbitrary output vectors in $\mathbb{R}^M$.

The MLP can be viewed as the standard/vanilla neural network but a range of alternative neural network architectures have been developed over the years. Specific architectures typically excel at specific tasks, and the development of suitable architectures for tasks such as image processing (convolutional networks) and natural language processing (recurrent and attention networks) has been instrumental in breakthroughs in those fields. While architecture development is the cornerstone of the field of deep learning, papers on DRL in inventory control typically adopt the standard MLP; there is clear future research potential here (cf. Section 4). The architecture determines the number and form of the parameters θ, but the numerical values of the parameters θ that provide the most desirable outcome are determined while *training* the network.

Training a network is conceptually similar to fitting the parameters of a regression model. It involves minimizing a *loss function*: a function that measures how well $\mathcal{N}_\theta(\cdot)$ matches the desired outcome. For instance, training a neural network to distinguish (images of) dogs and cats involves a training set of images, and the loss function measures the degree of errors made while categorizing these images. In general, the loss function guides a gradient-based iterative search for better values of the parameters θ. While the loss function guides this search, its minimization is not necessarily the end goal: for image recognition, for instance, the end goal is to correctly classify images of cats and dogs that were not part of the training set. In inventory control the end goal is to develop near-optimal replenishment policies that minimize corresponding costs.

The ability to adapt the parameters θ renders neural networks extremely versatile. In DRL, neural networks are employed to rep-

[2] Throughout the paper we adopt boldface notation for vectors.

[3] These equations are well-defined only if matrix and vector dimensions are such that all matrix-vector multiplications and vector-vector additions are well-defined.

[4] $\sigma : \mathbb{R}^K \to \Delta^K$ satisfies $\sigma(\mathbf{x})_i = \exp(\mathbf{x}_i) / \sum_{k=1}^K \exp(\mathbf{x}_k)$.

resent one or more key functions pertaining to the Bellman equations[5]:

1. In *value-based* methods neural networks represent the action values $q_\theta(s,a)$. As such they search for approximations of the optimal value functions and not the optimal policy;
2. In *policy-based* methods neural networks represent a policy $\pi_\theta(s)$. Hence they directly approximate the optimal inventory policy;
3. In *actor-critic* methods neural networks represent the policy $\pi_\theta(s)$ *and* the state-value function $v_\theta(s)$. This hybrid approach combines features of both value-based and policy-based methods by adding a value-based baseline to guide policy updates.

Consider for instance a state space $\mathcal{S} \subset \mathbb{R}^N$ for some N and a finite action space $\mathcal{A}$, with $M = |\mathcal{A}|$. Any neural network architecture $\mathcal{N}_\theta : \mathbb{R}^N \to \Delta^M$, e.g. some MLP which sets $f^{[L]}(\cdot) := \sigma(\cdot)$, can map the state vectors in $\mathbb{R}^N$ to a stochastic policy vector in Δ^M, such that the probability to take action a for state $\mathbf{s}$ corresponds to the a'th entry in the vector $\mathcal{N}_\theta(\mathbf{s})$, or $\pi_\theta(a|\mathbf{s}) := (\mathcal{N}_\theta(\mathbf{s}))_a$. Then π_θ is a valid stochastic policy for a given set of parameter values θ. The training algorithm then seeks numerical values for all matrix and vector entries in θ such that π_θ is a good policy, i.e., one with low expected costs. In typical DRL algorithms, it is instrumental that the probabilities $\pi_\theta(a|\mathbf{s})$ are continuous and differentiable w.r.t. (the components of) θ, which explains the popularity of stochastic policies in DRL.

In a similar way, the action-value function $q_\theta(s,a)$ can be represented by the neural network, except that $\mathcal{N}$ maps from $\mathbb{R}^N$ to $\mathbb{R}^M$ (instead of to Δ^M), by using the identity function for activation in the final layer. A standard value function $v_\theta(s)$ can be represented by a neural network from $\mathbb{R}^N$ to $\mathbb{R}^1$. Fig. 1 visualizes how a neural network can represent the action-values (panel a), the policy (panel b), or an actor-critic combination (panel c) with a stochastic policy (the actor) and the value function (the critic).

Example: For the classical lost-sales inventory system with leadtime τ, states are naturally expressed as τ-dimensional vectors $\mathbf{s} \in \mathbb{R}^\tau$: The first entry $(\mathbf{s})_1$ denotes inventory on hand, and remaining entries denote pipeline inventories. We assume that inventory is discrete, and let $\bar{x}$ denote the uniform upper bound on the optimal order quantity in any state (cf. Zipkin (2008)). Accordingly, $\mathcal{A} = \{0, 1, \ldots, \bar{x}\}$, i.e. $|\mathcal{A}| = \bar{x} + 1$. Then any network $\mathcal{N}_\theta : \mathbb{R}^\tau \to \Delta^{\bar{x}+1}$ can represent stochastic policies for the lost sales system with leadtime τ.

This representation hinges on specific assumptions regarding the MDP, e.g. that the action space is finite (we discuss these assumptions further in Section 4).

Reflecting on the sometimes complex structure of optimal inventory policies, one may wonder whether MLPs or other neural networks and their simple matrix multiplications could ever represent something as complex as a near-optimal (inventory) policy, even if matrices are carefully tuned. Actually, repeated affine transformations and non-linearities are more flexible than one may intuitively think: if neural networks can learn to distinguish between images of dogs and images of cats, then perhaps they can also develop good inventory policies. Besides, MLPs (even single layer neural networks) have been shown to be universal approximators, i.e., they are able to approximate continuous functions arbitrary well on compact sets (closed and bounded intervals) given a sufficient number of hidden neurons (Hornik, Stinchcombe, & White, 1989).

[5] While *deep learning* typically refers to state-of-the-art neural networks beyond the *standard* multi-layer perceptron, the term DRL is used to describe algorithms that seek to solve MDPs using any neural network architecture (including MLP).

Training a neural network to find a good MDP policy is rather different from training a neural network for classification purposes, such as image recognition. Instead of using a training set of images with labeled data, DRL makes use of (simulated or externally-collected) sequences of states, actions and rewards, often referred to as *trajectories*, to determine the loss function $\mathcal{L}(\theta)$. The loss function measures how well the neural net approximates the policy and/or value functions using observations in the trajectory and the output of the neural network (see Section 3.5 for more details). In every training iteration $k+1$, the values of the parameters θ_k are updated based on the gradient of the loss:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_{\theta_k} \mathcal{L}(\theta_k) \tag{5}$$

The learning rate $\alpha > 0$ may be optimized based on past gradient updates, e.g., ADAM (Kingma & Ba, 2015). The ultimate goal is to obtain good approximations of the optimal value functions and/or policy for state and action regions in- and outside those encountered in the training trajectories. A particular challenge of DRL is that when the loss function depends on θ, it changes over time, which renders DRL algorithms more difficult to tune compared to deep learning algorithms for classification purposes.

3. How to develop a DRL algorithm?

The choice of the most appropriate DRL algorithm for a specific optimization problem is not trivial. The most important design choice of a DRL algorithm relates to the design and purpose of the neural network. The neural network is the 'heart' of the DRL algorithm. As described above, DRL algorithms train neural networks by *learning* the numerical values of parameter set θ. That is, by iteratively updating the parameter set θ_k in every iteration k based on trajectories of actions, states and rewards, stored as tuples (s, a, c, s'), with c the cost or reward incurred when taking action a in state s. The end goal is to obtain a neural network that approximates the optimal policy $\pi_\theta(s) \approx \pi^*(s)$, the optimal value function $v_\theta(s) \approx v^*(s)$, and/or the optimal action-value function $q_\theta(s,a) \approx q^*(s,a)$.

Once the design of the DRL algorithm is fixed, the neural network can be trained. A key element to evaluate the (final) policy performance is to evaluate the model for different sets of hyperparameters. Hyperparameters are the parameters of the DRL algorithm that need to be set *prior* to the training process; that is, prior to the process of obtaining the weights and biases (parameter set θ) of the neural network. These hyperparameters are for instance the number of layers and their activation functions, the learning rate, or the length of experience buffers. The parameter set θ by contrast is updated *during* the learning process. Several search techniques exist to identify well-performing hyperparameter sets including manual search, grid search, random search or Bayesian optimization (for an in-depth discussion see e.g., Bengio, 2012; Bergstra, Yamins, & Cox, 2013; Snoek, Larochelle, & Adams, 2012). As different hyperparameter sets can result in different configurations of the neural network (and as such different policies), it is thus instrumental to compare different sets on their final policy performance. To evaluate one specific hyperparameter set, we must run the DRL algorithm, meaning that it learns values of parameter set θ that result in good policy performance. During the learning process, policy performance is occasionally evaluated through simulation. While evaluating, the weights of the neural network are not changed as that could change the policy suggested by the network. This process is repeated for multiple combinations of hyperparameter sets. The hyperparameter set and the learned weights of the neural network corresponding to it that result in the best policy performance are retained.

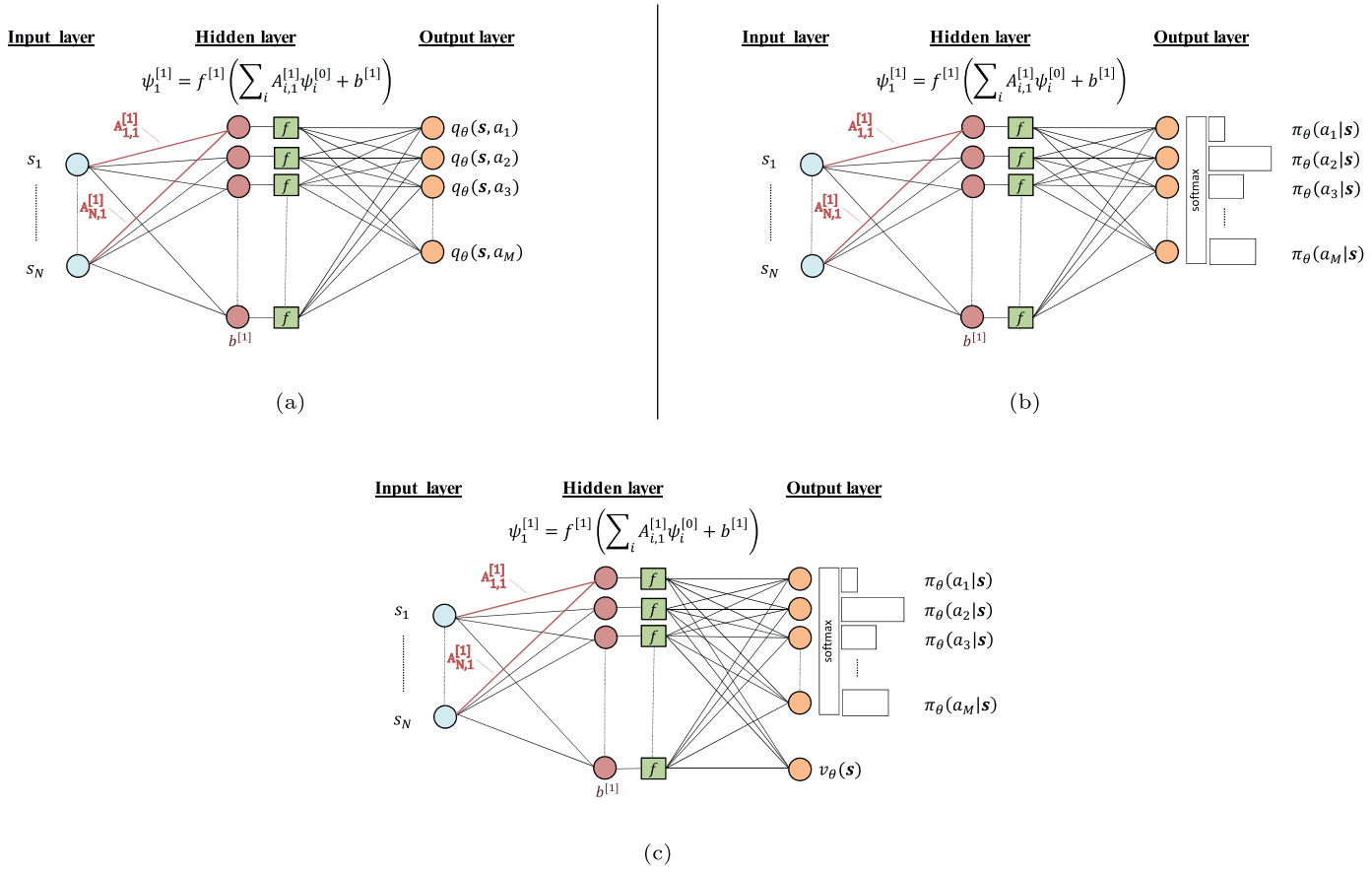

Fig. 1. DRL makes use of neural networks to represent either (a) the action-value functions $q_\theta(s, a)$, (b) the stochastic policy $\pi_\theta(s)$ or (c) an actor-critic combination with a stochastic policy $\pi_\theta(s)$ (the actor) and the value function $v_\theta(s)$ (the critic).

Table 1
Overview of influential DRL algorithms with their typical design specifications.

Design choices							
Algorithm	Role(s) of neural network	Offline	Online	Model-based	Model-free	Off-policy	On-policy
DQN	q-value estimation	✓	✓		✓	✓	
REINFORCE	policy representation		✓		✓		✓
A3C	hybrid/actor-critic		✓		✓		✓
TRPO	hybrid/actor-critic		✓		✓		✓
PPO	hybrid/actor-critic		✓		✓		✓
AlphaZero	hybrid		✓	✓		✓	✓

The evaluation of each design choice requires additional computation time. Due to the time-consuming process of selecting *and* evaluating the most appropriate designs it may take long to obtain decent performance and discourage interested users to start with DRL. To speed up this process we shed light on the main design choices in DRL algorithms and highlight what we believe is most impactful for inventory optimization. We do not aim to provide a tutorial on the various DRL algorithms (for that we refer the interested reader to e.g., François-Lavet, Henderson, Islam, Bellemare, & Pineau, 2018).[6] Some of these design choices are rather trivial for inventory applications, while other choices require more care because (like the role of the neural network) they have important repercussions. Table 1 summarizes the design choices of some well-known DRL algorithms.

[6] The code of many DRL algorithms is also available online, see for instance https://spinningup.openai.com, https://github.com/openai/baselines and https://stable-baselines.readthedocs.io.

3.1. Offline versus online

A first design choice relates to the question whether neural networks are trained based on previously collected data—in the form of (s, a, c, s') tuples—or on additional tuples collected during the learning process. This refers to offline versus online learning. In offline reinforcement learning algorithms (also referred to as batch reinforcement learning) no new tuples are collected during the learning process and the neural network is trained using a predefined set of tuples, e.g., based on a static data set with historical demands, orders and inventory levels. In settings where it is difficult to create new experience, offline learning may be instrumental. It enables turning (existing) data into decision-making without allowing exploration of new states and actions. To date, various technical challenges remain within the field of offline reinforcement learning and its full promise has yet to be realized (Levine, Kumar, Tucker, & Fu, 2020).

Online reinforcement learning algorithms, in contrast, create new tuples by interacting with the environment and explore new

regions of the state and action space as learning progresses. This interaction can be in the form of physical experiments that collect trajectories of states, actions and rewards, or it can be facilitated using a simulator. Such a simulator generates tuples by modelling the environment, i.e., the underlying MDP with its transition function between states and its reward function. Inventory settings typically assume the MDP is well-defined, although the uncertainty may have to be estimated from data. Nevertheless, they are typically amenable for simulation to generate orders, inventory levels and rewards. For instance, we may not know the true demand distribution but we may know the inventory balance equation: ordering one item more (compared to an existing tuple in the dataset) raises the pipeline inventory level of the transition state s' by one. Even when (demand) data are scarce, online algorithms may generate new observations by sampling from the empirical distribution, or any fitted distribution. Although this inherently results in reduced performance compared to using the true distribution, the same is true for all optimization models, including heuristic benchmark policies. Nonetheless, significant opportunities exist to make DRL more data-driven, as we discuss in the future research directions (see Section 4.1).

Example: Consider the lost sales problem, where the true inventory dynamics may be hard to recover. As we only observe the demands that have been fulfilled from inventory, the unmet demand (when inventory is depleted) remains unobserved. There may also be a correlation between the inventory level and demand. The modeler may wish to model these effects and build a simulation model (using a static data set) to support an online DRL algorithm. Alternatively, we may also train an offline DRL algorithm on the existing static dataset without building a simulator. In the latter case, however, the agent has no room to explore and create new data tuples. Whenever new data comes in, both online and offline methods may be re-trained on the expanded data set.

3.2. Model-free versus model-based

A second design choice relates to the question whether the agent (the decision-maker) uses the model (i.e., the simulator or MDP) to predict the impact of its actions on the future states and rewards *before* it takes an action. Before deciding on the order quantity, the agent may for instance compute the impact of various order decisions on subsequent inventory levels and costs. To do this, it may use the simulator to perform a Monte Carlo sampling to project future states and costs, or recursively compute projected inventory levels and costs using the transition and/or reward function of the MDP. The model may be fully known to the agent or may have to be learnt while training. In the latter case, the agent trains on estimations of the transition and reward functions of the MDP. The agent may for instance have access to the inventory balance equation to predict future states and costs, but may have to learn the demand distribution while observing samples, without having access to the true demand distribution. Algorithms that use the model when deciding on an action (for instance by looking ahead) are referred to as model-based algorithms.

Model-free algorithms cannot predict the future, and take actions without explicitly knowing the impact of their actions on the resulting states and rewards. Implicitly model-free value-based methods also learn (action-)value functions that represent the future costs. Yet, they cannot explicitly exploit the underlying model to predict future states and rewards to achieve better approximations: they merely interact with the environment and observe the resulting rewards and subsequent states. We note that, when the environment itself is a simulation model (as is often in the case in inventory management, in contrast to robotics that oftentimes learn in a physical world), also model-free algorithms make use of a model of the environment. In that case the difference is subtle: while both model-free and model-based algorithms use the model, model-free algorithms only use the model to simulate new sequences of states, actions and rewards but not for predicting future states and rewards when taking an action. The key trade-off for the modeler is to decide whether the (potential) performance improvement of exploiting the model offsets the additional computational effort *within* every simulated period to predict the impact of various decisions.

As can be observed in Table 1, many DRL algorithms are model-free. We attribute this to the fact that to date DRL has mostly focused on applications with environments that are too complex to model, such as robots or self-driving cars interacting in the real world. Besides, they are attractive as a general-purpose technology as the same algorithm can be applied to a variety of inventory problems without requiring too much domain knowledge. Model-based algorithms, in contrast, may benefit from less training iterations to develop a good policy as they look a few steps ahead and take this information into account when making a decision, albeit at the expense of increased computational effort and requiring more domain knowledge. Also in classical inventory control, state-of-the-art forward-looking heuristics and approximate dynamic programming policies consider a few steps ahead when placing orders. For example, a myopic lost sales policy chooses the order that minimizes the cost in the period in which the order arrives by recursively computing the expected inventory levels (Zipkin, 2008).

3.3. Exploration versus exploitation

The exploration exploitation trade-off is inherent to machine learning: agents need to choose between exploiting regions and actions that are known to perform well versus exploring new actions and regions. This is especially relevant when learning online where balancing well-performing actions against the exploration of new actions is important. Exploitation involves taking the currently best known action from each state while exploration implies we do not follow the recommendation of our neural network. Various approaches exist to explore new actions: ϵ-greedy approaches pick the best-known action with probability $(1-\epsilon)$ or a different (often random) action else; another exploration approach is to add 'noise' to the chosen action. The latter has the advantage to pick random actions that may be closer to the best-known action. Due to their probabilistic nature, stochastic policies inherently encourage exploration. To ensure that the policies continue to explore over time, it is in addition customary to penalize near-deterministic policies in the loss function by adding a so-called entropy term to the loss function.

Smart and careful exploration may be fruitful—especially when the algorithm is exploring good regions, and when taking bad actions may result in a sudden transition to poor regions. In inventory management it may take several periods to recover from one bad action. One excessive order, for instance, has to travel through the whole pipeline vector and may take multiple periods to be consumed by the demand. As such one could constrain the exploration to actions that lie within a specific distance from the actions proposed by the neural network. At the same time one may argue that in order to make the model more robust (i.e., such that it explores the entire state space), we may want to encourage more extreme actions to explore rare states or perhaps even impose uncertainty in e.g., demand or lead times. We discuss smart exploration in more detail as a future research avenue in Section 4.

3.4. On-policy versus off-policy

Another design choice refers to which policy is used to generate the tuples (s, a, c, s') that are used to train the neural network. *On-policy* algorithms update the weights of the neural network in each iteration based on tuples generated by the policy that is proposed by the current configuration of the neural network. *Off-policy* models can use experience from other policies, such as tuples generated from older versions of the neural network or tuples from a data set that was collected upfront, which it stores in an *experience buffer*. Notice the difference between on one hand, online versus offline learning and, on the other hand, on-policy versus off-policy learning. The former refers to the question *if* we can gather additional tuples: online algorithms generate new experience during the learning process, while offline algorithms do not; the latter refers to *how* online learning algorithms generate new tuples: on-policy methods follow and use the recommendations of the current neural network while off-policy methods also include experiences generated using different policies. Note that offline reinforcement learning is naturally off-policy as it uses data that is collected upfront according to some policy.

On-policy learning may benefit from theoretical convergence properties. The on-policy REINFORCE algorithm, for instance, is proven to converge to a local optimal policy (Williams, 1992). At the same time, the downside of on-policy learning is that it may get stuck in these local optima when only few data are used to feed the neural network. Off-policy algorithms are more "sample efficient", as they re-use the same tuples to train the neural network. Furthermore, when learning off-policy, the neural network can be trained with larger batches of simulation trajectories, as the experience buffer is orders of magnitudes larger than the trajectories sampled by on-policy algorithms. Feeding the neural network with more data allows to exploit the parallelization power of GPUs. This may provide more stability in the training process compared to on-policy learning, yet without any theoretical convergence properties.

3.5. The role of the neural network

The neural network is the core of any DRL algorithm. The most influential choice that distinguishes DRL algorithms is thus its role: for which component of the MDP do we wish to leverage the approximation power of the neural network. We separate between value-based, policy-based, and other hybrid approaches such as actor-critics in the following sections, and discuss their main pros and cons.

3.5.1. Value-based

Value-based methods, such as the seminal Deep Q-Network (DQN) use neural networks to approximate the optimal action-value functions. They do so by minimizing the value loss: the temporal difference between the estimated action-value function, $q_\theta(s, a)$, and the observed costs, c, after simulating one (or more) period(s), bootstrapped by the value function of the (last) state to which we transitioned, $\min_a q_\theta(s', a)$. A simple one-period temporal difference for the tuple (s, a, c, s') would thus be: $q_\theta(s, a) - \big(c + \gamma \min_a q_\theta(s', a)\big)$; the loss function to guide a gradient-based iterative search to update θ_k then follows using a mean squared error: $\mathcal{L}(\theta_k) = \big(q_{\theta_k}(s, a) - (c + \gamma \min_a q_{\theta_k}(s', a))\big)^2$. In practice, neural networks will be fed with many of these tuples, and minimize the mean squared error over all tuples. One major advantage of value-based methods is that they lend themselves better to off-policy learning, making them more sample efficient.

Yet, value-based methods search for approximations of the optimal value functions and not the optimal policy. Inherently they are thus set up to optimize the wrong objective. Eq. (4) may seem like a small step to convert the approximated value function into its policy but in general it is unclear how small approximation errors in the value functions impact the final policy performance. In fact, small errors often result in extremely poor policies. To make matters worse, value-based methods tend to be unstable and not converge well. Many value-based methods are off-policy and rely on bootstrapping of the value function based on the most recent version of the neural network. Using the approximation provided by the neural network itself to bootstrap may prevent the neural network to converge to stable approximations. In particular, the combination of function approximation (neural networks), off-policy learning and bootstrapping appears destructive. It is also referred to as the *deadly triad* (Sutton & Barto, 2018). Tsitsiklis & Van Roy (1997) have shown that this deadly triad makes learning unstable. Our lack of theoretically understanding the performance of value-based methods thus remains a limitation—and likely hampered adoption of these methods in the past. Early experiments in inventory management to combine neural networks and RL such as Van Roy, Bertsekas, Lee, & Tsitsiklis (1997) were successful but required manual selection of features.

To improve the performance of value-based methods, a number of heuristic techniques have been suggested. The DQN implementation of Mnih et al. (2015), for instance, uses algorithmic "tricks" such as experience replay and target networks to improve the empirical performance of DQN. In the wake of these breakthroughs, Oroojlooyjadid, Nazari, Snyder, & Takác (2021) shows how DQN performs well on the well-known supply chain Beer Game. We refer to the Rainbow implementation of Hessel et al. (2018) for a comparison of more algorithmic improvements, of which the potential in inventory control remains largely unexplored to date.

Value-based DRL methods are most fruitful when sample efficiency is important, for instance, when data is scarce or gathering new data is difficult or unreliable. Alternatively, policy-based methods may be preferred, which we discuss next.

3.5.2. Policy-based

Policy-based methods directly approximate the optimal inventory policy by minimizing the policy loss $\mathcal{L}(\theta_k)$ in each update k of the neural network. This policy loss is defined as the expected cumulative (discounted) cost obtained during a trajectory (of length T) while following policy π_{θ_k}: $\mathcal{L}(\theta_k) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t c'_t \mid \pi_{\theta_k}]$ with c'_t the cost incurred when taking action a_t in state s_t at time step t of the trajectory. Note that for policy-based methods the discount factor may be set to $\gamma = 1$, in contrast to (unmodified) value-based methods where the action values would inflate to infinity. In essence, our goal is thus to minimize the expected cost of the policy. Similar to value-based methods, the parameters θ_k of the neural network are updated in the direction of the gradient. The policy gradient theorem states that the gradient of the policy loss is the expectation of the product of the expected cost and the gradient of the log of the policy (see Marbach & Tsitsiklis, 2001; Sutton, McAllester, Singh, Mansour et al., 1999):

$$\nabla_{\theta_k} \mathcal{L}(\theta_k) = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_{\theta_k} \log \pi_{\theta_k}(a_t \mid s_t) \left(\sum_{t=0}^{T} \gamma^t c'_t\right)\right]. \tag{6}$$

This result is elegant as it allows us to use the sampled tuples without losing convergence guarantees. Perhaps counter-intuitively, we do not require the steady-state distribution nor any dynamics of the model (such as the exact transition or rewards functions) to compute the gradient of the neural network. The REINFORCE algorithm (Williams, 1992) for instance, directly follows from the policy gradient theorem. These vanilla policy gradient methods suffer, however, from high variance among the trajectories, i.e., the observed costs of parallel trajectories may vary significantly. Various expressions exist that effectively reduce this variance, such

as subtracting value-based baselines dependent on states we transition to, as is typically done in actor-critic methods. We discuss these methods next and refer to Schulman, Moritz, Levine, Jordan, & Abbeel (2015b) for a more in-depth discussion on variance reduction.

As the gradient update is directly computed using the policy, policy-based methods have better convergence properties in terms of policy performance (i.e., they converge to a local minimum of the cost function); a desired property for inventory control. Unfortunately, the policy gradient theorem no longer holds when tuples generated by older policies are used. In fact, we know this would introduce 'bias', potentially shifting the desired update in the wrong direction. For this reason many policy-based algorithms are on-policy and suffer from weaker sample efficiency.

3.5.3. Actor-critics and other hybrid techniques

Actor-critic methods are hybrid approaches that leverage multiple benefits of both value-based and policy-based methods. For instance, actor-critic algorithms add a value-based baseline to policy-based methods to reduce the variance caused by relying on sampled trajectories (Mnih et al., 2016). The increased stability of adding the baseline of the critic, however, goes at the expense of adding an additional value loss function compared to pure policy gradient methods. This comes at additional computation requirements. The A3C actor-critic models used by Gijsbrechts, Boute, Zhang, & Van Mieghem (2019) show good performance in lost sales, dual sourcing and multi-echelon inventory management, thereby demonstrating that DRL resembles a general purpose technology. They do highlight the computational burden to find good hyperparameters while many training runs still result in poor policies.

To cope with these shortcomings, different metrics may be used to update the policy in a smarter way, often resulting in better (sometimes theoretically justified) convergence, albeit at the expense of a more expensive computation in each training iteration. We specifically refer to Kakade (2001) for a discussion on natural gradients that may result in convergence to the optimal policy—despite being computationally very intensive or even intractable in practice. Kakade & Langford (2002) shows how conservative policy iteration (when restarting from past states is allowed) converges to approximately optimal policies. (Note how the latter implies the algorithm is model-based, as we discussed in Section 3.2.) They even provide an explicit bound when the policy update is a mixture of the old and new policy. The trust region policy optimization (TRPO) algorithm of Schulman, Levine, Moritz, Jordan, & Abbeel (2015a) is directly inspired by Kakade & Langford (2002). It also improves monotonically in every training iteration by using a hard constraint to keep the updates to the new policy within a specified range (measured using the KL-distance).

The above methods do not always work well in practice due to the excessive computational need in every training step. More empirical approaches were proposed following the aforementioned works, such as clipping the distance between the old and new policy in the proximal policy optimization (PPO) algorithm of Schulman, Wolski, Dhariwal, Radford, & Klimov (2017). The PPO algorithm was successfully applied in the joint replenishment inventory problem by Vanvuchelen, Gijsbrechts, & Boute (2020). The computational requirements could be further improved. For instance, rather than starting from scratch we may start from a well-known heuristic (or a heuristic currently in use by a company) and aim to improve monotonically on this policy as in Kakade & Langford (2002). Soft constraints may also be used to penalize deviations from the heuristic. A similar teacher student framework was proposed by Nazari, Jahani, Snyder, & Takác (2019), which may readily be applied in inventory settings. We elaborate further in the future research avenues.

Cutting-edge algorithms improve on typical architectures. The neural network architecture used in AlphaZero, for instance, strongly resembles that of actor-critic methods by combining a neural network that outputs both a value function and a policy. In addition to using a complex neural network including a combination of convolutional and residual networks, AlphaZero also uses self-play, in the form of a Monte Carlo Tree Search (MCTS) to improve the decisions of the actor.

4. Avenues for future research

The application of DRL in inventory management is novel and largely unexplored (apart from a few exceptions discussed in the previous section). In what follows we highlight a number of research avenues that may facilitate the adoption of DRL in practice.

4.1. DRL for data-driven inventory control

Traditional inventory control models assume the distribution of the uncertainty, such as the demand or lead time, to be known. To implement these models, a pre-processing step is required by first estimating or forecasting the uncertainty, before the inventory policy can be optimized. Instead of separating the estimation and optimization, the abundance of (big) data available today gave rise to a new, data-driven research stream within operations, in which both steps are no longer made sequentially, but integrated. Prominent works such as Mišić & Perakis (2020) and Simchi-Levi (2014) call for further research within data-driven decision-making. The advent of machine learning is deeply entangled with the availability of lots of data, but the use of data in DRL lags behind its supervised learning counterpart. Most (if not all) papers that explore DRL in inventory management focus on models where the uncertainty is assumed to be known by the modeler, neglecting the impact (and potential) of using data. Data-driven decision making is clearly underexplored in DRL. We believe DRL may become a complimentary tool in data-driven decision making and envision the following potential avenues to use DRL in data-driven inventory control:

Leverage big data to enrich the state space: A key benefit of DRL is that it allows to capture large state spaces. Therefore we could consider augmenting the state space with additional features that are traditionally used for estimation. For instance, we may add historical demands or lead times and exogenous data such as weather, news events, competitor activity or traffic information in the state space. Without explicitly forecasting the (demand or lead time) uncertainty, the DRL algorithm may learn to include these features in the training of the neural network and the corresponding optimization of the inventory policy.

Leveraging these big data does raise a few additional challenges. For instance, the DRL algorithm might require a model to interact with and it is not directly obvious how the model can generate samples when it only has access to historical data without understanding the (conditional) probability distributions between all state and random variables. This is essentially the core of data-driven optimization: integrating the estimation in the optimization by avoiding that small forecast errors propagate into the final policy performance. If significant data is available we may circumvent the need to generate new samples and use an offline learning approach by exclusively sampling from (a part of the) historically observed data samples, without the need to generate additional experience. In most cases, however, we want the DRL algorithm to learn online by exploring unobserved states and actions. Sampling new data points requires some additional engineering. For instance, we may generate new samples by combining bootstrapping from historically observed samples (and their feature values) and engineering new samples (for instance by adding random noise on

existing samples). This noise can be set depending on how closely the observed state variables resemble an existing data point in the dataset (i.e., we may assume the random variables more closely track values of nearest neighbours in the dataset). Alternative to manually engineering samples, we may also let the DRL algorithm come up with data points to make the resulting policy more robust, as we discuss next.

Develop more robust policies: An alternative data-driven approach that has gained popularity in inventory control is robust optimization, in which the worst-case cost is minimized rather than the expected cost. The worst case may be defined based on the available data samples. We refer to Mamani, Nassiri, & Wagner (2017), Bertsimas & Thiele (2006) and the references therein for a discussion on the use of robust optimization in inventory control but note that the construction of the worst case, which is an essential component of robust optimization, is typically left to the discretion of the modeller. As DRL algorithms are notoriously unstable and non-robust, an interesting avenue to explore is the inclusion of the worst case in the training of the DRL algorithm in order to make the policy more robust to uncertainties. For instance, we may use an additional neural network (or DRL algorithm) that outputs the uncertainty and tries to negatively impact the other one: the adversarial DRL algorithm would observe the same state but instead of deciding how much to order, it would output the random variables. This is very similar to how Generative Adversarial Networks (GANs) are being used in supervised learning to make algorithms more robust with respect to small deviations in the inputs (Goodfellow et al., 2014). That is, the adversarial network has permission to slightly modify or create new samples (for instance pictures) to fool the initial predictor. The initial network will then be retrained and learn to make more robust predictions due to the additional challenge imposed by the presence of the adversarial network. This approach can be used to make the policies more robust with respect to limited data availability, inaccurate data, or extreme events that increase the level of uncertainty.

Develop policies for dynamic environments: The current DRL applications in inventory control assume stationary environments, such as stationary demand or lead times. Many real-life problems are inherently dynamic, think of for instance the product lifetime where the demand distribution evolves over time. In particular with respect to data-driven decision-making this problem becomes apparent as data may quickly become outdated in fast-changing environments (for instance, in fast-fashion). Future research may focus on developing DRL algorithms that can cope with non-stationary environments. Despite not being explored, this does not seem impossible as specific configurations of neural networks have been developed specifically to identify patterns and trends. These network configurations may also be appropriate to develop replenishment policies for SKUs in dynamic environments such as products with short product life cycles or with strong seasonality. One of the main modelling challenges will be to adapt the neural network to these changes. For instance, in case of a rapid demand growth, the output layer of the neural network may have to be adjusted such that the action space stays in line with the increasing demand realizations.

4.2. Blending numerical and analytical approaches to optimize inventory policies

As we have discussed in Section 2, inventory policies can be optimized through MDPs in an analytical or numerical way, and using exact or approximate methods. To date, the application of DRL to inventory problems falls exclusively in the latter category of numerical and approximate methods, aimed at finding good inventory policies. While we acknowledge this has value in an initial research stage, we believe there is a lot of untapped potential of DRL to integrate the aforementioned roles by leveraging and obtaining new structural results using the inherently numerical approach of DRL.

Leverage analytical results to improve DRL performance: Inventory control is a forerunner in understanding value function structures and related properties for stylized problems. An interesting avenue for further research is the integration of these results into algorithmic design in order to steer the DRL algorithm into the right direction. It has for instance been shown that the value function of the lost sales inventory problem is $L^\natural$-convex (Zipkin, 2008). This theoretical property has been used to develop excellent heuristics, such as the Best Weighted Bounds policy (Hua, Yu, Zhang, & Xu, 2015) in dual sourcing or approximate dynamic programming policies that use linear programming to fit the parameters of an $L^\natural$-convex function to approximate the value functions of dual sourcing with stochastic supply and lost sales (Chen & Yang, 2019; Sun, Wang, & Zipkin, 2014). In a similar way, we may inject these analytical properties into DRL algorithms by shaping the value function through smart elimination of sub-optimal actions or states, for instance, by penalizing the loss function when its shape does not resemble the desired structure. An additional advantage of exploiting these theoretical properties is the potential to find and compare against clever performance bounds, potentially even proving worst-case performance bounds of the DRL algorithm itself. For dual-source inventory models, one could for instance use well-established single-sourcing optimality results to bound state or action spaces, or as a general upper performance bound.

Extract simple heuristic policies from DRL outputs: Instead of leveraging analytical results to improve the numerical performance of DRL, the numerical policies developed by DRL may also spur the development of new, simpler heuristic policies that can be analytically characterized, and are thus easier to implement. A novel approach to convert numerical results into analytic insights is proposed by Bravo & Shaposhnik (2020). They leverage exact numerical methods to find the optimal value functions for a range of MDPs, and subsequently use the output as input to a machine learning method to extract analytic insights into the structure of the optimal policy for a range of problem domains: inventory management, queuing admission control, multi-armed bandits, and revenue management. This exciting result thus uses exact numerical methods to derive new analytic insights. Similarly, we could use the approximate numerical methodology of DRL to learn the structure of well-performing solutions, which may lead to new heuristic policies for challenging problems that have until now resisted exact or approximate analysis. Once their structure is learned, these heuristic policies can be implemented without the need to train any neural network.

Understand and improve performance of heuristic policies: The OR/OM community has a strong focus on knowledge building and benchmarking these new approaches against existing policies. Accordingly, most early studies that apply DRL to inventory management benchmark DRL performance against the state-of-the-art problem-specific heuristic policies (see e.g., Gijsbrechts et al., 2019; Vanvuchelen et al., 2020). As DRL matures, we may reverse this process and use DRL to benchmark the performance of existing (or new) heuristic policies, and understand why—or when—these heuristics perform better or worse. Demonstrating that a new heuristic matches (or beats) well-designed general-purpose DRL benchmarks may provide evidence that the heuristic performs well. Comparing their performance against the DRL policies may also identify combinations of problem parameters (e.g., certain cost environments) where the heuristic performs poorly. We may then drill down one level and identify the states where the DRL agent takes substantially different actions from the heuristic. This may in turn allow us to formulate an improved heuristic.

Explain DRL policies: Neural network policies are flexible and performant, but notoriously difficult to interpret. This sharply contrasts with the often highly intuitive character of policies obtained via analytic methods, such as for instance base-stock or constant order policies. In addition to developing more intuitive policies from the complex output of neural networks similar to Bravo & Shaposhnik (2020), we may also develop models that additionally explain *why* an action is proposed. A vast field exists on explaining and interpreting the output of AI models (see also Gilpin et al., 2018). When models not only output actions, but also provide managers the intuition behind the action, the adoption of DRL in practice will be fostered.

4.3. Scalability of DRL algorithms

Since DRL relies only on a general MDP formulation (and basically *any* practical inventory problem can be characterized by an MDP), DRL algorithms may *in principle* deliver good policies for any inventory model for which it is difficult to derive simple, well-performing heuristics. This observation is an important driver for the excitement about DRL for inventory control: DRL could bring inventory research closer to practical implementations.

Apart from providing proof of concept that DRL can be applied as a general-purpose technology to classic, yet intractable inventory problems, Gijsbrechts et al. (2019) also highlighted various limitations of the current state-of-the-art DRL algorithms for inventory management: they require extensive tuning to identify the best performing set of hyperparameters of the neural network. This is computationally very expensive. Moreover, the inventory models considered in these early studies are still restricted to *stylized models* in the sense that they consist of a single SKU and one or a few locations. This is in contrast with the complexity of typical practical inventory systems such as for instance the one considered in Kranenburg & Van Houtum (2009). They consider a realistic case study of a lost sales inventory system at the manufacturer ASML, who produces photolithography machines that are used in the production of computer chips. Their setting consists of 19 warehouses/locations, 27 machine groups, and 1451 SKUs. To facilitate the implementation of DRL for inventory control with a realistic scale, it is clear that we should go beyond the current state-of-the-art of DRL applications. Various future research avenues pertain to this ambition:

Better DRL training algorithms: Researchers in OR/OM may continue to draw from innovations within the active field of DRL. From Vanvuchelen et al. (2020), for instance, it appears that PPO requires less tuning compared to A3C as employed by Gijsbrechts et al. (2019), though it should be noted that the algorithms were not compared on the same problem. In particular, model-based algorithms may be tailored to inventory research with the option to generalize well across problems, just like MuZero learned to play multiple games at master class level (Schrittwieser et al., 2020), or one may even develop DRL algorithms specifically tailored to the theoretical characteristics of *inventory problems* obtained after decades of in-depth research, as we discuss next.

Improve training performance: To speed up the training process, we may leverage the structural knowledge of the heuristics that are known to perform well. The transfer of such external expert knowledge when training a neural network is studied in *imitation learning*, which is a popular research field in supervised learning as it may significantly improve training performance and reduce computational requirements. To date, this field remains largely unexplored in DRL and especially in inventory control. One approach to leverage known heuristics or structural knowledge would be to start the training process from a neural network that trains on samples generated by a well-performing heuristic. This might improve training performance compared to starting from a randomly initialized neural network. These ideas are studied in a sub-field called *behaviour cloning*, the simplest form of imitation learning. Behaviour cloning is a supervised learning problem in which the neural network learns a policy based on a training set of states and expert decisions. The *teacher student framework*, in addition, enables the decision maker to generate efficient policy improvements (student) guided by 'teacher' advice. The teacher can for instance be a well-performing heuristic or structural knowledge about the inventory policy. This framework ensures better guidance of the learning process towards well-performing state/action regions and policies (see e.g., Nazari et al., 2019 for an example outside inventory control). Rewards may also be altered or shaped, known as *reward shaping* to speed up the training process and to make training more stable, without losing performance (Ng, Harada, & Russell, 1999). De Moor, Gijsbrechts, & Boute (2021) recently successfully applied reward shaping to stabilize the learning process of DQN for a perishable inventory problem, while their model occasionally outperforms the best-known heuristics.

Extend to large action spaces: While there appear to be no *fundamental* limits to the size of state spaces that DRL can address, this stands in contrast with the limitations for action spaces. Neural networks are mappings from $\mathbb{R}^N$ to $\mathbb{R}^M$, where in many models M directly corresponds to the number of actions, i.e., $M = |\mathcal{A}|$. Larger values of M (say $M = 1000$) lead to neural networks with many parameters, which are more difficult, if not impossible, to train. Without modifications, this implies that DRL can only be used for MDPs/inventory models with discrete action spaces, and $|\mathcal{A}|$ small enough. In inventory control, large action spaces may arise when no upper bound can be proven on the size of orders or when this upper bound is very large, and also when orders take values in a continuous interval. Discretizing the set of potential orders may lead to theoretical performance loss.

An even more pressing concern arises in inventory models where *multiple* actions need to be taken simultaneously, e.g., when there are multiple SKUs or when inventory at a location must be allocated for shipment to multiple down-stream locations.

Example: When managing inventory for multiple products, one has to decide how much to replenish for each item. Let I denote the number of items and $\bar{x}_i$ the upper bound on the optimal order quantity for item i in any state, and assume for simplicity $\bar{x}_i = \bar{x}$ for all $i \in \{1, \ldots, I\}$ and zero lead times. The latter implies that the state space is I-dimensional, consisting of the inventory levels of each product. The number of actions (i.e., the order quantity for each item), and the resulting number of output nodes of the neural network in a standard DQN or policy-based algorithm then equals $|\mathcal{A}| = (\bar{x} + 1)^I$ and the action-value or policy functions can be represented by a neural network, $\mathcal{N}_\theta : \mathbb{R}^I \to \mathbb{R}^{(\bar{x}+1)^I}$.

As the number of output nodes of the neural network increases exponentially in the number of items, standard DRL algorithms become less suitable for inventory problems with a large action space. Most inventory models that are important in practice feature such simultaneous actions (see e.g. Kranenburg & Van Houtum, 2009), and overcoming this limitation is a foundational avenue for future research into increasing scalability of DRL. We next discuss various ideas that have appeared in literature for dealing with multi-dimensional action spaces.

The policy (and as such the action space) may be both discrete and continuous. The action space of continuous policies are typically represented in a neural network by letting the output represent the first moments of a pre-specified distribution from which the action is sampled. Moreover, action spaces may be multi-dimensional, as is often apparent in robotics where multiple joints need to be steered simultaneously. These techniques are rather new and have potential for multi-product or multi-sourcing inventory problems in particular, and multi-dimensional action spaces in

general. As such, continuous action spaces may be useful also for discrete inventory problems with large action spaces as continuous policies require less output nodes compared to discrete policies. Until now no such models have been successfully trained within inventory control; it is not clear how representing the order size in a single output node will affect the ability to effectively train the neural networks.

Another approach that may be taken is to train a separate neural network for each action that needs to be taken. Apart from issues with regards to the computational demands of training many neural networks, a challenge here is that actions may be interdependent (e.g., products are transported in the same truck or when inventory is allocated over multiple downstream locations). Several approaches have been proposed to overcome this (see for instance Baker et al., 2020; Kaynov, 2021; Pirhooshyaran & Snyder, 2020). To reduce the computational demands of training various neural networks, Pirhooshyaran & Snyder (2020) propose in addition to share the weights for the first few layers of the neural network. Alternatively, multi-agent DRL (Baker et al., 2020), in which multiple agents optimize individual or (partly) shared objective functions may be a good fit to tackle large, multi-dimensional action spaces. We may, for instance, model products as individual agents that have their own service level penalties but share inventory or transportation space.

Another approach to deal with simultaneous actions is to leverage symmetry in the problem. In inventory control, this idea was originally proposed by Van Roy et al. (1997) in the context of a symmetric one-warehouse multi-retailer inventory setting. For that case, it suffices to train a single neural network to make all allocation decisions, as those decisions will be the same because the retailers are identical. While complete symmetry is not so common in practice, partial symmetry is much more common and perhaps there are ways to similarly use that. Think for instance of the replenishment process of items that only differ in cost parameters and/or lead times. In the field of deep learning, *transfer learning* uses previously trained neural networks for some task as a starting point to learn a different (potentially related) task. These dissimilarities are by no means limited to the examples previously given. Instead, these tasks can differ in several aspects of the MDP: state/ action space, reward function or the transition dynamics. To date, only Oroojlooyjadid et al. (2021) employs this technique for inventory control. They use a neural network trained for one agent in the Beer supply game and use transfer learning to quickly adapt the network to other agents and other parameter settings. Their results are promising as they were able to reduce training time by one order of magnitude. A final approach to cope with large action spaces, is *meta-learning*. This is essentially learning how to learn. The ultimate goal is learning a model that can efficiently adapt (i.e. only require a small amount of training iterations) to solve new test tasks by training the model on a variety of different training tasks. We may for instance use meta-learning when a firm has a large amount of products: rather than training a DRL algorithm for each product individually, we may train one meta-learning across a set of SKUs such that with only minor additional training time it can be trained to the specific SKUs.

DQN and other *value-based* DRL methods are less suitable for dealing with large state spaces. However, under the banner of approximate dynamic programming (ADP), a vast range numerical and approximate methods have been proposed that utilize linear function approximators to approximate the value function, see e.g., Powell (2007). Those approaches fall outside the scope of this review paper because they do not use deep neural networks, but their adoption of linear value function approximations enables them to exploit problem structure to efficiently deal with high-dimensional action spaces.

5. Conclusion

Despite being coined as a promising technique to tackle complex sequential decision-making problems for which analytical or exact numerical solution methods fall short, applications of DRL algorithms in inventory control remain rather scarce. A significant effort is needed to construct and train DRL algorithms, making it notably harder to use than its supervised deep learning counterpart, often without performance guarantees. We shed light on how neural networks may effectively be employed as an approximate numerical approach to optimize inventory problems that are modeled as MDPs. In addition to shedding light on the essential design choices of DRL algorithms we highlight new research avenues which have not been explored to date.

As an operations community, and especially within inventory control, we have invested strongly in theory building—one of the strengths of our field. While it is common to use results from pioneering works from decades ago, the field of AI, and particularly machine learning, is newer, less structured and often less supported theoretically. We believe that as DRL for inventory control becomes a more established field, it must also have focus on theory building. Blending the numerical approach of DRL with analytical results may be fruitful for inventory control.

As DRL is extremely versatile to tackle diverse problems without prior knowledge, we hope our paper may inspire beyond the boundaries of inventory control within operations management and management science.

References

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., et al. (2020). Emergent tool use from multi-agent autocurricula. *Proceedings of the 8th international conference on learning representations (ICLR)*.

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society, 60*(6), 503–515.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade* (pp. 437–478). Springer.

Bergstra, J., Yamins, D., & Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th international conference on machine learning* (pp. 115–123).

Bertsekas, D. P. (1987). *Dynamic programming: Deterministic and stochastic models*. Englewood Cliffs, NJ: Prentice-Hall.

Bertsekas, D. P. (2019). *Reinforcement learning and optimal control*. Belmont, MA: Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Bertsimas, D., & Thiele, A. (2006). A robust optimization approach to inventory theory. *Operations Research, 54*(1), 150–168.

Bravo, F., & Shaposhnik, Y. (2020). Mining optimal policies: A pattern recognition approach to model analysis. *INFORMS Journal on Optimization, 2*(3), 145–166.

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.

De Moor, B. J., Gijsbrechts, J., & Boute, R. N. (2021). Reward shaping to improve the performance of deep reinforcement learning in inventory management. Available at SSRN: https://doi.org/10.2139/ssrn.3804655.

Dulac-Arnold, G., Mankowitz, D., & Hester, T. (2019). Challenges of real-world reinforcement learning. arXiv:1904.12901

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning, 11*(3–4), 219–354.

Garychl (2018). Applications of Reinforcement Learning in Real World. Retrieved April 13, 2021, from https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12.

Gershgorn, D. (2018). There's only been one AI breakthrough. Retrieved April 13, 2021, from https://qz.com/1419346/ai-has-had-just-one-breakthrough-says-kai-fu-lee.

Gijsbrechts, J., Boute, R., Zhang, D., & Van Mieghem, J. (2019). Can deep reinforcement learning improve inventory management? Performance on dual sourcing, lost sales and multi-echelon problems. Available at SSRN: https://doi.org/10.2139/ssrn.3302881.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., & Kagal, L. (2018). Explaining explanations: An overview of interpretability of machine learning. In *Proceedings of the 5th ieee international conference on data science and advanced analytics (DSAA)* (pp. 80–89).

Goldberg, D. A., Katz-Rogozhnikov, D. A., Lu, Y., Sharma, M., & Squillante, M. S. (2016). Asymptotic optimality of constant-order policies for lost

sales inventory models with large lead times. *Mathematics of Operations Research, 41*(3), 745–1160.
Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. In *Proceedings of the 27th international conference on neural information processing systems* (pp. 2672–2680).
Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., et al. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the 32nd aaai conference on artificial intelligence* (pp. 3215–3222).
Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*(5), 359–366.
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.
Kakade, S., & Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th international conference on machine learning* (pp. 267–274).
Kakade, S. M. (2001). A natural policy gradient. *Advances in Neural Information Processing Systems, 14*, 1531–1538.
Karlin, S., & Scarf, H. (1958). Inventory models of the Arrow–Harris–Marschak type with time lag. In *Studies in the mathematical theory of inventory and production* (pp. 155–178)). Stanford University Press.
Kaynov, I. (2021). Deep Reinforcement Learning for Asymmetric One-Warehouse Multi-Retailer Inventory Management. Available at: https://research.tue.nl/en/studentTheses/deep-reinforcement-learning-for-asymmetric-one-warehouse-multi-re.
Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *Proceedings of the 3rd international conference on learning representations (ICLR)*.
Kranenburg, A., & Van Houtum, G.-J. (2009). A new partial pooling structure for spare parts networks. *European Journal of Operational Research, 199*(3), 908–921.
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097–1105.
Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv:2005.01643
Lu, Y., & Song, J.-S. (2005). Order-based cost optimization in assemble-to-order systems. *Operations Research, 53*(1), 151–169.
Mamani, H., Nassiri, S., & Wagner, M. R. (2017). Closed-form solutions for robust inventory management. *Management Science, 63*(5), 1625–1643.
Marbach, P., & Tsitsiklis, J. N. (2001). Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control, 46*(2), 191–209.
Martagan, T., Krishnamurthy, A., Leland, P. A., & Maravelias, C. T. (2018). Performance guarantees and optimal purification decisions for engineered proteins. *Operations Research, 66*(1), 18–41.
Mišić, V. V., & Perakis, G. (2020). Data analytics in operations management: A review. *Manufacturing and Service Operations Management, 22*(1), 158–169.
Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., & Harley, T. et al. (2016). Asynchronous methods for deep reinforcement learning. arXiv:1602.01783
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.
Nazari, M., Jahani, M., Snyder, L. V., & Takác, M. (2019). Don't forget your teacher: A corrective reinforcement learning framework. arXiv:1905.13562
Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th international conference on machine learning* (pp. 278–287).
Oroojlooyjadid, A., Nazari, M., Snyder, L. V., & Takác, M. (2021). *A deep q-network for the Beer Game: deep reinforcement learning for inventory optimization*. Manufacturing & Service Operations Management.
Pirhooshyaran, M., & Snyder, L. V. (2020). Simultaneous decision making for stochastic multi-echelon inventory optimization with deep neural networks as decision makers. arXiv:2006.05608
Powell, W. B. (2007). *Approximate dynamic programming: Solving the curses of dimensionality*. Hoboken, NJ: John Wiley & Sons.
Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. Hoboken, NJ: John Wiley & Sons, Inc..
Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., et al. (2020). Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature, 588*(7839), 604–609.
Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015a). Trust region policy optimization. arXiv:1502.05477
Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. arXiv:1506.02438
Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv:1707.06347
Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of Go without human knowledge. *Nature, 550*(7676), 354–359.
Simchi-Levi, D. (2014). OM forum-OM research: From problem-driven to data-driven research. *Manufacturing and Service Operations Management, 16*(1), 2–10.
Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems, 25*, 2951–2959.
Sun, P., Wang, K., & Zipkin, P. (2014). *Quadratic approximation of cost functions in lost sales and perishable inventory control problems*.
Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press Cambridge.
Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy gradient methods for reinforcement learning with function approximation.. In *Proceedings of the 12th International Conference on Neural Information Processing* (pp. 1057–1063).
Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8*, 257–277.
Tesauro, G. (1994). TD-Gammon, a self-teaching Backgammon program, achieves master-level play. *Neural Computation, 6*(2), 215–219.
Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control, 42*(5), 674–690.
Van Roy, B., Bertsekas, D. P., Lee, Y., & Tsitsiklis, J. N. (1997). A neuro-dynamic programming approach to retailer inventory management. *Proceedings of the 36th IEEE Conference on Decision and Control*.
Vanvuchelen, N., Gijsbrechts, J., & Boute, R. (2020). Use of proximal policy optimization for the joint replenishment problem. *Computers in Industry, 119*, 103239.
Veinott, A. F. (1966). The status of mathematical inventory theory. *Management Science, 12*(11), 745–777.
Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*(3), 229–256.
Zipkin, P. (2008). Old and new methods for lost-sales inventory systems. *Operations Research, 56*(5), 1256–1263.