Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management*

Bram J. De Moor[a,*], Joren Gijsbrechts[b], Robert N. Boute[a,c]

[a] *Research Center for Operations Management, KU Leuven, Naamsestraat 69, Box 3555, 3000 Leuven, Belgium*
[b] *Católica Lisbon School of Business and Economics, Palma de Cima, 1649-023 Lisbon, Portugal*
[c] *Technology and Operations Management Area, Vlerick Business School, Vlamingenstraat 83, 3000 Leuven, Belgium*

Abstract

Deep reinforcement learning (DRL) has proven to be an effective, general-purpose technology to develop 'good' replenishment policies in inventory management. We show how transfer learning from existing, well-performing heuristics may stabilize the training process and improve the performance of DRL in inventory control. While the idea is general, we specifically implement potential-based reward shaping to a deep Q-network algorithm to manage inventory of perishable goods that, cursed by dimensionality, has proven to be notoriously complex. The application of our approach may not only improve inventory cost performance and reduce computational effort, the increased training stability may also help to gain trust in the policies obtained by black box DRL algorithms.

Keywords: Inventory, Perishable inventory management, Deep reinforcement learning, Reward shaping, Transfer learning

1. Introduction

Deep reinforcement learning (DRL) is a subbranch of machine learning that makes use of artificial neural networks to find policies for sequential decision-making problems. By formulating the problem as a Markov Decision Process (MDP), the neural nets serve as approximators of the corresponding value or policy functions. A DRL agent 'learns' these value or policy functions by simulating actions, interacting with the environment and incurring rewards. This is referred to as the training process of the neural network. Despite the fact that it does not guarantee an optimal solution to the MDP, this learning-by-doing method circumvents the curse of dimensionality that plagues exact methods such as dynamic programming.

*To appear in European Journal of Operational Research
*Corresponding author
Email addresses: bram.demoor@kuleuven.be (Bram J. De Moor), jgijsbrechts@ucp.pt (Joren Gijsbrechts), robert.boute@kuleuven.be (Robert N. Boute)

DRL gained popularity due to a number of recent successes to find winning policies in (computer) games. Notable milestones include mastering seven Atari 2600 games (Mnih et al., 2015); the victory of the DRL algorithm AlphaGo over former world champion Lee Sedol in the complex game of GO (Silver et al., 2016); the creation of AlphaZero, a better and more general-purpose algorithm than AlphaGo (Silver et al., 2018); and the development of MuZero, achieving superior performance in GO while also excelling in other games like chess by learning the rules of the game while training (Schrittwieser et al., 2020). The superhuman performance on these games provided a stepping stone for DRL applications in other fields, ranging from robotics (Haarnoja et al., 2019) to autonomous vehicles (Kiran et al., 2021).

Inspired by these success stories, DRL has also found its way to inventory control. It provides a numerical and approximate method to solve complex inventory problems for which no analytical solution exists to date, or for which exact numerical methods such as value iteration quickly become intractable (Boute et al., 2021). Oroojlooyjadid et al. (2020) demonstrate that a deep Q-network (DQN) can find near-optimal policies for the well-known beer distribution game. Gijsbrechts et al. (2020) use the Asynchronous Advantage Actor Critic (A3C) algorithm and show that it is able to develop near-optimal policies for three classic inventory problems: dual sourcing, lost sales and multi-echelon inventory management. Vanvuchelen et al. (2020) use proximal policy optimization (PPO) to tackle the joint replenishment problem, showing that the algorithm approaches the optimal policy structure and occasionally outperforms other well-performing heuristics.

One of DRL's shortcomings is its sample inefficiency: a vast amount of simulation experience is needed until the network starts developing reasonable policies. This comes at a high computational cost (Thompson et al., 2020). For instance, the hardware requirements needed to train AlphaGo Zero (a newer version of AlphaGo), cost around 25 million USD (Silver et al., 2017; Gibney, 2017). One way to learn more efficiently is to drop the conviction that a DRL algorithm should learn from scratch. Instead, one could incorporate domain knowledge embedded in well-performing heuristic policies into the training algorithm. After all, the inventory field contains a vast amount of domain knowledge thanks to decades of research. Current DRL approaches in inventory management do not yet tap into this pool of knowledge; instead they primarily focus on the strength of DRL as a general-purpose technology. We add to this research stream by demonstrating that analytical results can be instrumental to improve the performance of DRL, thereby making further advances to the use of DRL in inventory control.

Transfer learning is a machine learning method that starts training from prior knowledge instead

of learning from scratch. Most transfer learning algorithms transfer low-level knowledge, like value functions or the weights of a neural net, by exploiting pre-trained neural networks that were used for a similar problem. Policy transfer methods use knowledge from other 'teacher' policies. One way to do so is to manipulate the rewards, which an RL agent observes while learning a good policy, based on the teacher policy. The technique of manipulating rewards is referred to as *reward shaping* (Skinner, 1938). Whereas reward shaping is typically used to tackle sparse reward signals, we use reward shaping for policy transfer to encourage a DRL agent to learn from a teacher heuristic. When the DRL agent takes a certain action and the system transitions to a new state, it gives rise to a reward, which we modify based on the recommended action of a teacher heuristic. This incentivizes the agent to act similar compared to the action proposed by the teacher. We conceptualize the resulting agent-environment interaction under reward shaping in Figure 1.

To illustrate: consider a large grid world with obstacles in which a DRL agent gets a reward if it reaches the upper most right corner. Suppose that we know a good, not necessarily optimal, route to reach the target and use this route as the teacher policy. By positively rewarding actions corresponding to this teacher policy, we can not only speed up the learning process, but also potentially learn better policies. Alternatively, one can also manipulate the rewards negatively when the agent deviates from the teacher policy. Brys et al. (2015) show in small experiments how reward shaping in reinforcement learning can extract knowledge embedded in a teacher policy, as long as the agent does not get distracted from its main goal. Indeed, it is important that the optimal policy of the modified MDP, by changing its rewards, remains identical to the optimal policy of the original MDP, a condition called *policy invariance*. Ng et al. (1999) introduce potential-based reward shaping as a necessary and sufficient condition to achieve policy invariance. Although theoretical proofs do not yet exist (and may not even be possible to obtain) when neural nets are used to approximate the value function, our numerical results empirically suggest that policy invariance properties under potential-based reward shaping are also maintained under *deep* reinforcement learning. We will further elaborate on potential-based reward shaping in Section 4.2.

In this paper we demonstrate how potential-based reward shaping can be used to leverage domain knowledge embedded in heuristic inventory policies to improve the learning process of DRL for inventory control. While this has many potential applications, we implement this technique on the inventory management of perishable goods, for which the optimal policy is intractable. We use two teacher policies to shape the rewards in a deep Q-learning (DQN) algorithm: (1) the celebrated base-stock policy that, despite being sub-optimal, is widely used in practice thanks to its

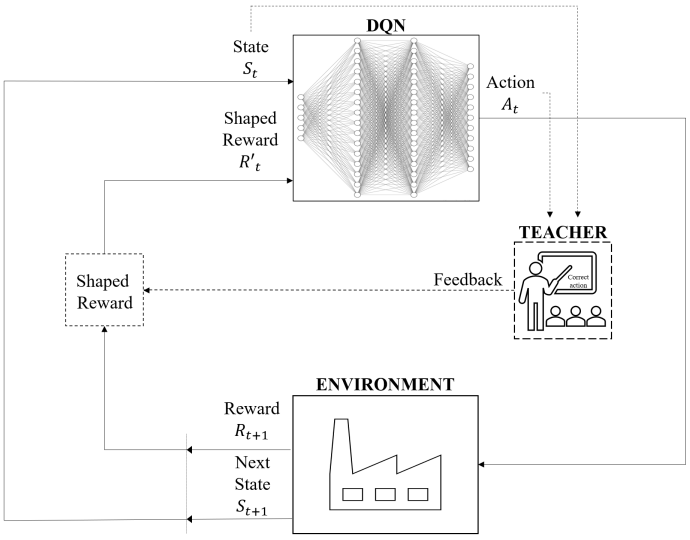

Figure 1: Agent-environment interaction in reinforcement learning using a teacher to shape rewards. The dashed lines show the interactions that differ from regular reinforcement learning. Each time step, an agent, in this case a DQN, observes a state and takes a certain action based on that. The next time step, given that action, the system transitions to a new state and a reward is generated. The reward observed by the agent is manipulated by a teacher that gives feedback on the action taken.

simplicity; and (2) a modified base-stock policy with estimates of waste (BSP-low-EW), the best-known heuristic to date for perishable goods inventory management (Haijema and Minner, 2019). Our numerical study shows that this reward-shaping technique can improve performance compared to a DQN without reward shaping. When the teacher policy outperforms the 'unshaped' DQN (and when the optimality gaps are not negligibly small), the shaped DQN can also surpass its teacher's performance. When the teacher is inferior compared to unshaped DQN, reward shaping does not improve policy performance obtained by DQN. However, even with a relatively poor teacher, we observe that reward shaping results in more efficient learning per training episode, especially in the first part of the training process. Thus, less training episodes (and thus less computational effort) will be needed to achieve the same performance level when using reward shaping. Furthermore, we find that shaping makes a DRL model more robust with respect to the used hyperparameters: we observe that when a good teacher policy is available, policies tend to converge more easily to reasonably good policies compared to unshaped models without tuning. Lastly, we observe that reward shaping stabilizes the training process. That is, the average cost performance and its variability are substantially lower when reward shaping is used. This may increase the trust in DRL algorithms to obtain reliable performance.

Note that our transfer learning approach is different from the one of Oroojlooyjadid et al. (2020). They use a pre-trained neural network to speed up learning. In contrast, we transfer knowledge from existing heuristic policies that are known to perform (reasonably) well. It shows how analytical results can benefit numerical methods by potentially improving performance as well as stabilizing and accelerating the training process. Furthermore, we provide a visual analysis of different heuristics and policies found by the (shaped) DQN which may spur follow-up research to develop new heuristic policies, hence benefiting the perishable inventory community. Lastly, we contribute by providing an application of potential-based reward shaping in *deep* RL. As most applications are limited to mere RL, we thus expand the boundaries of potential-based reward shaping, which may stimulate more methodological (convergence) proofs—although this may be tedious or even impossible.

From a practical point of view, we believe that our work may support the introduction of DRL in companies to improve their inventory practices by employing the currently used policies as teachers. Compared to unshaped DRL, shaping can lead to computational savings, a potential decrease in replenishment costs, and a model that is less dependent on tuning. Moreover, the use of known policies as teachers and the resulting increase in stability may help companies to gain trust in the policies obtained by the black box DRL algorithms.

While we only consider the conventional perishable inventory problem, we believe our results extend to other inventory management problems, or operations at large, where good heuristics are available. For instance, the fact that our approach continues to work well for longer product lifetimes hints at good performance in non-perishable inventory management. We leave extensions to other problem settings (such as including non-stationarity) for future research.

2. A review of perishable inventory models

As defined by Nahmias (2011): "A perishable good is a product that has constant utility up until an expiration date, after which its utility drops to zero." The complexity of a product's perishability renders its optimal inventory management notoriously difficult. Indeed, the optimal replenishment policy is a complex function of both the inventory position *and* the age distribution of the inventory. Consequently, even under a stringent zero lead time assumption, finding the optimal replenishment policy requires the solution of an $(m-1)$-dimensional dynamic program, with m being the lifetime of the perishable good. Interestingly, this is the case both when considering backlogging and lost sales (Nahmias, 1975b; Fries, 1975). Solving this dynamic program with traditional approaches

like value or policy iteration quickly becomes intractable for increasing values of m, a phenomenon which is referred to as the curse of dimensionality (Bellman, 1957).

For positive lead times, $L > 0$, finding the optimal replenishment policy becomes even more challenging as the optimal order quantity then not only depends on the m-dimensional age distribution of the inventory on hand, but also on the $(L-1)$-dimensional vector of units in transit. Deriving the optimal solution thus boils down to solving a dynamic program with an $(m+L-1)$-dimensional state space—a computational agony (Williams and Patuwo, 1999; Haijema and Minner, 2019).

There are a number of asymptotic cases of m for which the optimal inventory policy is known. When $m = 1$, the problem effectively reduces to a one-period problem for which a newsvendor solution is optimal. Bulinskaya (1964) shows that the optimal order quantity is in this case independent of the orders in transit. When $m = \infty$, the problem reduces to a non-perishable one. Assuming zero lead time, we know that the base-stock policy, which orders up to a certain level S whenever inventory falls below this level, is optimal, both under a backlogging and lost sales assumption (Arrow et al., 1951). Bu et al. (2020) reinforce this by presenting a decreasing optimality gap of the base-stock policy for increasing m. When there is a positive lead time, the optimal policy for a backlogging system (without perishability) remains a base-stock policy, but it becomes a complex function of the units in transit under lost sales (Karlin and Scarf, 1958).

The complexity of the optimal policy stimulated research on approximate ordering policies. With a simulation study, Nahmias (1975a) finds that the base-stock policy is a good heuristic as it performs reasonably well and is easy to implement. We confirm these findings, but find that DRL easily outperforms a base-stock policy, even in the absence of reward shaping. Cohen (1976) and Chazan and Gal (1977) extend the results of Nahmias (1975a) by deriving the optimal order-up-to level analytically. Haijema (2013) adjusts the well-performing base-stock policy by introducing an extra parameter that caps the number of units that can be ordered. The capped base-stock policy performs better because it diminishes the expected cost of perishing.

As conventional base-stock policies only consider the inventory position and ignore the age of units in stock and transit, a number of modified base-stock policies have been proposed that include some notion of age distribution. Haijema et al. (2007) divide the inventory in new and old items depending on a parameter, and use two order-up-to levels: one for the total inventory and one for new inventory. A similar approach is taken by Duan and Liao (2013). They introduce an old-to-new inventory ratio and adjust the order depending on the relative position of this ratio compared to a threshold parameter. Haijema and Minner (2019) express the difference between old and new

inventory by weighing the units in each age category and using this weighted inventory in a classic base-stock policy. Broekmeulen and van Donselaar (2009) add estimated waste calculations to reach the desired inventory position when a positive lead time is present. The estimated waste is equal to the number of units in inventory that are expected to perish during the lead time, assuming mean demand in every period. Haijema and Minner (2019) exploit this idea by adding estimated waste calculations to a variety of modified base-stock policies. Such forward looking policies that consider estimated waste perform well. However, they are computationally more intensive compared to simple base-stock type policies, as they require to solve a recursion in every period.

In a large simulation study, Haijema and Minner (2019) introduce a modified base-stock policy with estimated waste, and crown this new policy, called BSP-low-EW, the best performing heuristic in practically all settings considered. This policy uses two order-up-to levels, S_1 and S_2, depending on the inventory position, y_t, relative to a threshold, b. The order placed in period t, q_t, under a BSP-low-EW policy is defined by:

$$q_t = \begin{cases} [S_1 - \alpha y_t + EW_t]^+ & y_t < b \\ [S_2 - y_t + EW_t]^+ & y_t \geq b, \end{cases}$$

with:

$$\alpha = 1 - \frac{S_2 - S_1}{b},$$

and EW_t the estimated waste during lead time at time t, assuming the demand in every period is deterministic, equal to the mean.

A specific aspect of perishable inventory systems is the issuing policy. As the optimal replenishment policy is a function of the units at different age levels, it matters which items are sold first. The two extreme examples are: sell the oldest items first, referred to as first-in-first-out (FIFO), or sell the youngest items first, referred to as last-in-first-out (LIFO). Pierskalla and Roach (1972) have shown that FIFO outperforms LIFO by large. Intuitively, this is because FIFO reduces the number of units that will perish soon, which is in the interest of reduced perishability costs. However, the decision between picking the youngest or oldest item is typically made by the consumer, who tends to follow a LIFO issuing policy, especially when the expiration date is known (Cohen and Pekelman, 1978). We have tested both issuing policies in our paper. Our numerical results confirm the findings of Pierskalla and Roach (1972).

We will use both the conventional base-stock (given its wide use) and the BSP-low-EW policy (the best heuristic to date) as teacher policies to improve the performance of our DRL algorithm. We

will also use these policies to benchmark the performance of DRL. We contribute to the (perishable) inventory literature by showing how existing teacher policies can improve the (learning) performance of DRL to develop better replenishment policies. We also analyze the structure of the different heuristic policies, which may inspire further research to develop new heuristic policies.

3. Model formulation

We consider a periodic-review, single item, single echelon, perishable inventory problem with stochastic demand and a fixed, known, positive lead time L. We assume unmet demand is lost and each item has a fixed, known lifetime m. We consider both FIFO and LIFO issuing policies.

Each period t, we decide on the number of units to order, q_t, after having observed the inventory on-hand $I_t = (q_{t-L} = i_1, i_2, ..., i_m)$ and the orders in transit $Q_t = (q_{t-1}, q_{t-2}, ..., q_{t-L-1})$, with i_j the units in inventory of age $j \in \{1, 2, ..., m\}$. The order placed at time $t - L$ is added to the inventory at the beginning of period t. The pipeline vector transitions by 'shifting' the orders as follows: $Q_{t+1} = (q_t, q_{t-1}, q_{t-2}, ..., q_{t-L-2})$. The inventory on hand evolves based on the demand observed in period t, d_t, and the issuing policy (FIFO or LIFO). Let $i_{j,t}$ be the units in inventory of age j at time t. Assuming a FIFO-issuing policy and lost demand, the inventory vector in the next time step $I_{t+1} = (i_{1,t+1}, i_{2,t+1}, ..., i_{m,t+1})$ is described by:

$$i_{j,t+1} = i_{j-1,t} - \left[d_t - \sum_{k=j}^{m} i_{k,t} \right]^+ \qquad \forall \quad j = 2, 3, ..., m.$$

Assuming a LIFO-issuing policy, the inventory vector transitions according to:

$$i_{j,t+1} = i_{j-1,t} - \left[d_t - \sum_{k=1}^{j-1} i_{k,t} \right]^+ \qquad \forall \quad j = 2, 3, ..., m.$$

For $j = 1$ we get, independent of the issuing policy:

$$i_{1,t+1} = q_{t-L-1}.$$

We can formalize this inventory problem as an MDP with state $s_t = (Q_t, I_t)$ and action $a_t = q_t$. The state space $\mathcal{S}$ is thus $(m + L - 1)$-dimensional and the finite action space is one-dimensional, described by the set of feasible order quantities $\mathcal{A} = \{0, 1, 2, ..., q_{max}\}$, with q_{max} the maximum order quantity. Positive left-over inventory at the end of the period incurs a positive unit holding cost (c_h), order costs are linear with a per unit ordering cost (c_o), unmet demand is lost incurring

a per unit lost sales penalty (c_l) and units that perish incur a per unit cost of perishing (c_p). The cost incurred at time step t, when taking action a_t in state s_t, can thus be defined as:

$$c_t(s_t, a_t) = c_o a_t + c_h \left[\sum_{k=1}^{m} i_{k,t} - d_t - \epsilon_t \right]^+ + c_l \left[d_t - \sum_{k=1}^{m} i_{k,t} \right]^+ + c_p \epsilon_t,$$

with $\epsilon_t = [i_{m,t} - d_t]^+$ the number of units that perish in period t, and $[\cdot]^+ = \max[0, \cdot]$ the positive part operator.

The goal is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$, mapping each state to an action, that minimizes the expected discounted value of future costs. The value function of being in state s_t following policy π is equal to the expected discounted value of future costs:

$$V^{\pi}(s_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k c_{t+k}(s_{t+k}, a_{t+k} | a_{t+k} \sim \pi(s_{t+k})) \right],$$

with $0 < \gamma < 1$ the discount factor. The value function of the current state s_t is recursively related to the value function of the next state s_{t+1}. By solving the well-known Bellman equations (Bellman, 1957), the optimal value function can be found for each state:

$$V^*(s_t) = \min_{a_t \in \mathcal{A}} \left[\mathbb{E}\left[c_t(s_t, a_t) \right] + \gamma \sum_{s_{t+1} \in \mathcal{S}} \left[\mathbb{P}(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \right] \right],$$

where $\mathbb{P}(s_{t+1} | s_t, a_t)$ is the probability of ending up in state s_{t+1} after taking action a_t in state s_t.

Analogously, one can define the optimal Q-value of each state-action pair as follows:

$$Q^*(s_t, a_t) = \mathbb{E}[c_t(s_t, a_t)] + \min_{a_{t+1} \in \mathcal{A}} \left[\sum_{s_{t+1} \in \mathcal{S}} \left[\mathbb{P}(s_{t+1} | s_t, a_t) Q(s_{t+1}, a_{t+1}) \right] \right], \tag{1}$$

which is dependent on state *and* action and only differs in the first element from the value function.

The optimal policy $\pi^*(s)$ is then easily obtained from these Q-values:

$$\pi^*(s) = \arg\min_{a \in \mathcal{A}} Q^*(s, a).$$

DQN circumvents the curse of dimensionality, specific to dynamic programming methods, by approximating the optimal Q-values with an artificial neural network (Mnih et al., 2015). We will apply this method to the perishable inventory problem described above. Because it is conventional in computer science literature to 'maximize rewards' instead of 'minimizing costs' in inventory management, we will define the reward as the negation of the cost. We will also formulate the

reward function as a function of s_t, a_t and s_{t+1}, instead of using a reward function based only on current state s_t and action a_t. This aligns our notation with the notation commonly used in the reward shaping literature.

4. Policy transfer in DQN using reward shaping

4.1. Deep Q-network (DQN)

DQN approximates the optimal Q-values with a neural network by simulating experience and updating the neural network's weights using this simulation experience (Mnih et al., 2015). In essence, a neural network is a powerful function operator, consisting of several layers which each contain a number of nodes and activation functions. We refer to Boute et al. (2021) for a more extensive discussion of the use of neural networks for inventory control, but merely state here that a neural network produces an output (in the output layer) based on an input (fed to the input layer) and the weights of the connections between the nodes of two adjacent (hidden) layers. The neural network(s) form the core of the DQN algorithm: they convert any state (input) into its set of Q-values for different actions (output). We denote $Q(s, a; \theta)$ as the output of the neural network, with θ the weights of the neural network. The objective of the DQN algorithm is to find the parameter set θ that minimizes the error between its approximations of the Q-values, $Q(s, a; \theta)$, and the optimal ones, $Q^*(s, a)$, as defined in (1).

Since the neural network outputs Q-value estimates for each state, the size of the input layer is equal to the dimension of the state space; in our case, the $(m + L - 1)$-dimensional inventory vector. The size of the output layer equals the size (not the dimension) of the action space. The neural network improves its approximations of the optimal Q-values by collecting experiences and replaying them, a mechanism called *experience replay*. As the agent takes an action a_t in state s_t, resulting in an immediate reward r_t and a next state s_{t+1}, an experience is created, formally defined as $e_t = \langle s_t, a_t, s_{t+1}, r_t \rangle$. This experience is stored in a replay buffer, $B_t = \{e_{t-M}, e_{t-M+1}, ..., e_t\}$, which keeps track of the last M experiences and discards older ones. After a fixed number of K experiences are collected, a random sample of N experiences is uniformly drawn from B_{t+K} and stored in a mini batch (note that M, K and N can be tuned by the user to optimize performance). This sampled set of N experiences is used to update the weights of the neural network to minimize the loss by applying a gradient descent update using the well-known Adam optimizer (Kingma and Ba, 2017). The total loss at iteration i is defined as:

$$\mathcal{L}_i(\theta_i) = \sum_{e \sim U(B)} \left[\left[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_i^-) \right] - Q(s_t, a_t; \theta_i) \right]^2, \tag{2}$$

with $e \sim U(B)$ denoting that N experiences are drawn uniformly from the replay buffer B; θ_i the weights of the neural network at iteration i; and θ_i^- the weights of the *target network*. A target network is an additional neural network of which the weights are not updated during every training iteration. Instead, they are updated every τ training episodes by putting them equal to the main network's weights at that specific training iteration. The target network estimates the unknown optimal Q-values, which are used to quantify the error of the approximation of the Q-values provided by the main network. This process of using an approximation as a target for another approximation is called *bootstrapping*. Bootstrapping with a separate target network has been shown to substantially stabilize the learning process (Mnih et al., 2015).

4.2. Reward shaping

Reward shaping adjusts the rewards of the MDP to bias the learning algorithm towards desired behavior. In its most general form, a shaped reward has the form:

$$R' = R + F, \tag{3}$$

with $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ the expected reward function of the original MDP, which we augment by the expected reward shaping function $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ resulting in the modified expected reward function $R' : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$. Instead of learning a policy based on R, we now consider R' and optimize the policy that maximizes the discounted sum of the adjusted rewards R'. Reward shaping is an effective tool to steer agents towards desired behavior, as proven by an abundance of success stories (Mataric, 1994; Dorigo and Colombetti, 1994; Devlin et al., 2011; Efthymiadis and Kudenko, 2013; Brys et al., 2014). However, when used improperly, reward shaping can adjust the optimal policy of the MDP, leading to undesired behavior. Randløv and Alstrøm (1998) provide a textbook example in their paper on how to learn an AI-agent to drive a bike to a certain target location using reinforcement learning. To help the agent, additional rewards are given for moving towards the target. Instead of going for the 'finish reward', the only reward in the original MDP, the proposed shaping function pushes the agent into riding circles for ever, collecting much more artificial rewards than it would get when reaching the desired location. This poses the question on how much a reward can be manipulated, without changing the optimal policy of the original problem. This condition is called *policy invariance*.

Ng et al. (1999) show how rewards should be shaped to guarantee policy invariance. They introduce a potential function $\Phi_{\mathcal{S}} : \mathcal{S} \mapsto \mathbb{R}$ that maps every state to an arbitrary value. Preferred states (e.g., states which we visit following a desired policy), are given a high value, and vice versa. The potential function is inspired by the *potential*, used in physics, denoting the potential energy of a physical body in a certain state, relative to a reference state. Using this potential function, Ng et al. (1999) define a potential-based reward shaping function $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ as:

$$F(s_t, a_t, s_{t+1}) = \gamma \Phi_{\mathcal{S}}(s_{t+1}) - \Phi_{\mathcal{S}}(s_t). \tag{4}$$

Ng et al. (1999) prove that the optimal policy of the shaped MDP, $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, R + F \rangle$, is also optimal in the unshaped MDP, $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, R \rangle$; with $\mathcal{S}$ the state space, $\mathcal{A}$ the action space, $\mathcal{T}$ the next-state transition probabilities, R the original reward function and F the reward shaping function. Interestingly, policy invariance holds for any arbitrarily chosen $\Phi_{\mathcal{S}}$, as long as the reward shaping function F is in the specific potential-based reward shaping form (4).

Wiewiora et al. (2003) extend the results of Ng et al. (1999) by using potential functions based on both states and actions, that is, $\Phi_{\mathcal{SA}} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. This new potential function gives more specific information to steer an agent towards desired behavior. Indeed, we can now give a value to state-action pairs, instead of only to states. Wiewiora et al. (2003) introduce two potential-based reward shaping functions, using the potential function $\Phi_{\mathcal{SA}}$, i.e., look-ahead and look-back reward shaping respectively. Look-ahead reward shaping uses the potential function of the next state-action pair (s_{t+1}, a_{t+1}), similar to Ng et al. (1999), whereas look-back reward shaping uses the potential function of the previous state-action pair (s_{t-1}, a_{t-1}) in a potential-based reward shaping function $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. We use the latter, defined as:

$$F(s_t, a_t, s_{t-1}, a_{t-1}) = \Phi_{\mathcal{SA}}(s_t, a_t) - \gamma^{-1} \Phi_{\mathcal{SA}}(s_{t-1}, a_{t-1}). \tag{5}$$

In general, one can not easily inject (5) into (3) as the former is dependent on the policy being followed. In other words, using this reward shaping function, we cannot simply express a new MDP with the adjusted reward function, $F + R$, and thus also not claim policy invariance theoretically. A notable exception to this case is when the followed policy is stationary (see also Wiewiora et al. (2003) for a theoretical justification). The reason is that, in expectation, the potential function of the previous state-action pair can be expressed when the policy being followed is fixed. Thus, under a stationary policy π we can define $R' = R + F^{\pi}$, where F^{π} denotes the dependency on the stationary policy being followed, and claim policy invariance. In our work, the DQN algorithm

is updated in every training such that the followed policy is not stationary (and hence we cannot theoretically claim policy invariance). Yet, we empirically confirm in Section 5 that shaped DQN algorithms occasionally outperform the teacher policies, hinting that policy invariance may in fact hold (even though such a result remains unproven to date).

We use look-back potential-based reward shaping introduced by Wiewiora et al. (2003) by defining the potential function $\Phi_{\mathcal{SA}}$ according to a teacher policy. We propose the use of well-known inventory heuristics as teacher policies to steer the DQN algorithm. Denote $a^{\dagger}(s)$ as the action taken by the teacher policy in state s. We define the potential function $\Phi_{\mathcal{SA}}$ as follows:

$$\Phi_{\mathcal{SA}}(s,a) = -k\,|a^{\dagger}(s) - a|,$$

with $k \in \mathbb{R}^{+}$ an arbitrarily chosen parameter to enforce the teacher's advice more strongly. Thus, the larger the difference between the action a taken in state s and the teacher's action $a^{\dagger}(s)$ in state s, the lower the potential function will be. By using this potential function in the look-back potential-based reward shaping function of Wiewiora et al. (2003), see (5), we thus indirectly promote actions which are in line with the teacher policy.

We acknowledge that Devlin and Kudenko (2012) introduce dynamic potential functions based on states, which is extended by Harutyunyan et al. (2015) to include dynamic potential functions based on both states and actions. This approach gradually learns a potential function with a separate learning method, by feeding it artificial rewards provided by a teacher. We tested this approach by learning a potential function with a separate DQN, using the exact same hyperparameters as the main DQN, with the only exception of a lower learning rate. We omit the in-depth description of the method and results as our dynamic shaping implementation did not seem to have an impact and generally converged towards unshaped DQN.

So far there is no proof available that when neural nets are used to approximate the value function, the obtained policies will converge to the optimal ones. As such, previous potential-based reward shaping literature has mainly focused on (tabular) reinforcement learning. We provide an application of potential-based reward shaping in *deep* reinforcement learning. Although we cannot make any theoretical claim, our results empirically suggest that policy invariance properties are maintained under *deep* reinforcement learning.

Experiment	μ	CoV	m	L	c_l	c_h	c_p	c_o	Issuing policy
1	4	0.50	2	1	5	1	7	3	LIFO
2	4	0.50	2	1	5	1	7	3	FIFO
3	4	0.50	2	1	5	1	10	3	LIFO
4	4	0.50	2	1	5	1	10	3	FIFO
5	4	0.50	2	2	5	1	7	3	LIFO
6	4	0.50	2	2	5	1	7	3	FIFO
7	4	0.50	2	2	5	1	10	3	LIFO
8	4	0.50	2	2	5	1	10	3	FIFO

Table 1: Eight parameter settings used to benchmark DQN with reward shaping against unshaped DQN and heuristics.

5. Numerical experiment

5.1. Experimental setup

We design different numerical experiments that each represent a specific perishable inventory problem. We first set the lifetime (the shelf life once an order arrives) of the perishable good $m = 2$ and extend to longer lifetimes in Section 5.3. All instances have a per unit cost of lost sales $c_l = 5$, a per unit holding cost $c_h = 1$, and a per unit order cost $c_o = 3$. The lead time L varies between 1 and 2. Per unit perishing cost c_p alternates between 7 and 10. Both FIFO and LIFO issuing policies are considered. Demand is gamma-distributed with a mean $\mu = 4$ and a coefficient of variation $CoV = 0.50$. The parameters of the different settings can be consulted in Table 1.

A limited, manual hyperparameter tuning resulted in adequate results in line with the benchmark policies. We did not perform any advanced search to further optimize the hyperparameters due to the high computational demands. Our analysis is thus conservative as better performance could be obtained if the hyperparameters were tuned more extensively.[1] However, as we will show, reward shaping reduces the sensitivity of the DRL performance with respect to the hyperparameters. Our manual search suggested the following hyperparameter values. The number of nodes in the input layer of the neural network equals the length of the vector describing the state. We thus have $(m + L - 1)$ input nodes. The number of nodes in the output layer is equal to the number of possible actions, $(q_{max} + 1)$; in all our experiments q_{max} is set to 10. Furthermore, we construct two hidden layers with 32 nodes each, ReLU activation functions and Dense connections. The output

[1]The interested reader is referred to Moolayil (2019) for an in-depth description of hyperparameter tuning.

layer has a Linear activation function, meaning that an output signal proportional to the input signal is created. To gather experiences, we play several 'games' of $K = 1,000$ periods and store experiences in a replay buffer of length $M = 20,000$. During each game of 1,000 periods, actions are selected based on an ϵ-greedy strategy: with a chance of $(1-\epsilon)$ we exploit the network's knowledge by taking the best action, and with a chance of ϵ we take a completely random action to explore the environment. The exploration parameter ϵ is set to 1 at the beginning of the first game and decays every game by multiplying it with 0.997, until a minimum of 0.01 is reached. After simulating 1,000 periods (one game), we add the 1,000 experiences to the replay buffer and sample a random batch of $N = 32$ experiences, calculate the loss as mean squared error (see (2)) using a discount factor $\gamma = 0.99$ and update the network weights using a learning rate of 0.001 and the Adam optimizer (Kingma and Ba, 2017). We define this as one training episode, which we repeat 2,000 times. Every 20 episodes (so every 20 $\times$ 1,000 periods), we update the weights of the target network. Appendix A provides a summary of all employed hyperparameters.

Each (shaped) DQN algorithm is initiated 50 times, i.e., 50 neural networks per experiment are trained with random weight initialization. The best-performing neural network of these 50 iterations is retained. We benchmark the performance of DQN, with and without reward shaping, against the base-stock and BSP-low-EW policy. The latter are also used as teacher policies to shape the rewards using the look-back potential-based reward shaping function, as defined in (5). All policies are evaluated based on the long-run average cost per period using Markov chain analysis. That is, we find the steady-state distribution and expected cost of each policy. For the reward shaping implementations, we set the parameter that determines the size of the shaping, k, to 50 as we have found through experimentation that this value generally gives good results. The experiments where $m = 2$ are sufficiently small such that optimal policies can be obtained with value iteration. In such small settings, tabular Q-learning would also suffice witout relying on neural nets. Yet, we chose to implement DQN as it allows scaling up to larger settings (see Section 5.3). At the same time it also explores the use of potential-based reward shaping in *deep* RL.

Each neural network was built, compiled and trained using Keras: a commonly used, high-level software library that runs on Tensorflow (Chollet, 2017). Training a model took around eight CPU hours on an AMD EPYC™ 7402 24-core processor, both for shaped and unshaped DQN, and there were no notable training time differences between experiments.

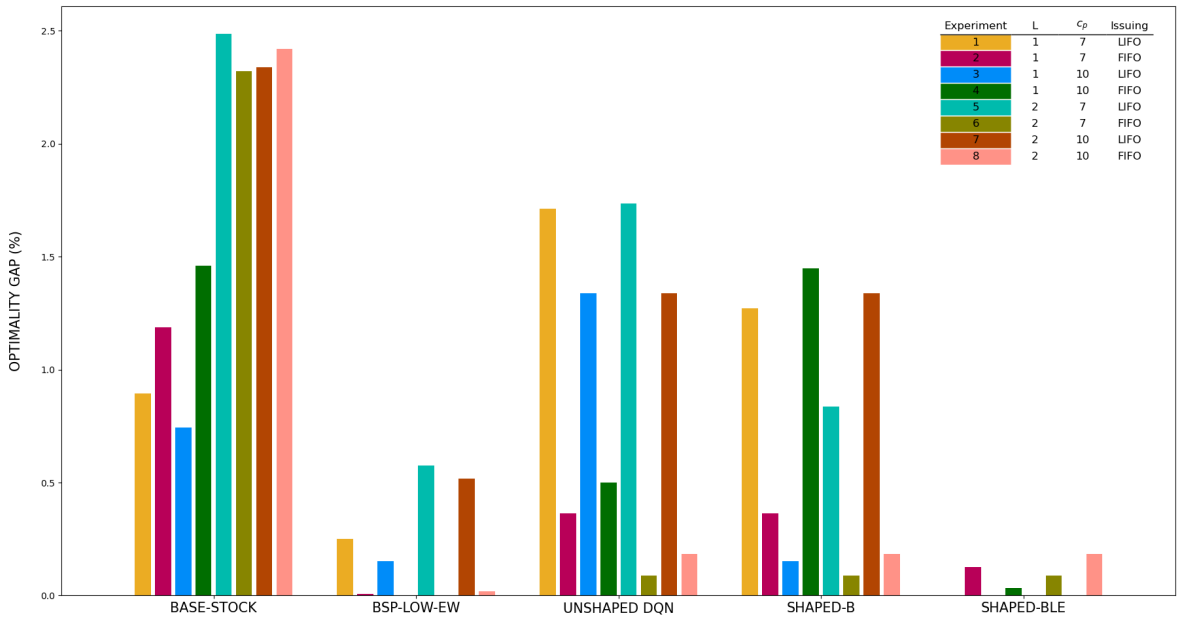

Figure 2: Optimality gaps of the teacher heuristics and the policies found by different DRL models (for $m = 2$).

5.2. Results for lifetime $m = 2$

Figure 2 displays the optimality gaps of the two teacher policies (base-stock policy and BSP-low-EW), the policy obtained by the unshaped DQN algorithm, and the two shaped DQN algorithms. Shaped-B and shaped-BLE refers to the policies obtained by shaping the DQN with the base-stock policy and the BSP-low-EW policy, respectively. When shaping with the base-stock policy as teacher, we see that shaped DQN improves on the teacher but shows no significant improvement over the unshaped DQN. This is because in most cases the unshaped DQN already performs better than the base-stock teacher policy. When a better teacher is used, such as the BSP-low-EW policy, we see that the shaped DQN clearly outperforms the unshaped counterpart. Furthermore, in four out of eight experiments where the BSP-low-EW policy performs sub-optimal, the student DRL algorithm outperforms the teacher, which is the best policy we know to date. In other words, potential-based reward shaping allows students to deviate and even improve on their teacher in search for the true optimal policy.

When the state space has two dimensions, we can visualize and interpret the different replenishment policies (as well as their steady state probabilities), as we do in Figure 3 for experiment 1 (LIFO) and 2 (FIFO). Experiment 3 and 4 are excluded from the visual analysis as they yielded similar results. For each heat map the x-axis represents the number of units in stock of age 2 and the y-axis the number of units in stock of age 1. The top row of each experiment visualizes the order

quantity of the respective policy given the number of units in stock of age 1 and 2. The bottom row provides the steady state probabilities of the respective policy, i.e., the probability of being in a state when following a certain policy. In general, we see that most policies tend to aim for states which have some units of age 1 in inventory, while keeping the units of age 2 low, in an attempt to avoid perishability. For the same reason we observe that orders will be higher when using a FIFO issuing policy compared to a LIFO policy.

In contrast to the base-stock policy (which only has one parameter), the BSP-low-EW policy has the flexibility to resemble the optimal policy to an acceptable degree in experiment 2 (FIFO). Yet, it does not manage to do so in experiment 1 (LIFO). This confirms our finding that the optimality gaps of BSP-low-EW are higher under LIFO. In both experiments we observe that unshaped DQN performs poorly and lacks the creativity to learn beyond a (semi-)constant order policy. Using shaping, however, we see that the DQN can be manipulated towards the optimal policy, such that their steady state probabilities closely resemble those of the optimal one.

Apart from improving the performance of the best trained neural network for each DRL model, we have also found that shaping improves the learning process: the average cost performance and variability is substantially lower compared to unshaped models. The top part of Figure 4 displays the average learning curve of the different DRL models in one experiment (see experiment 2 in Table 1), smoothed over 10 periods, visualized with 95% confidence intervals (we have observed similar training curves for the other experiments). One can clearly see that shaping lowers the variability and average optimality gap. In other words, when training one neural network, there is a higher probability of finding a 'good' policy with reward shaping compared to the unshaped model. That means that even if the best iterations of the Shaped-B models did not differ by much compared to those of the unshaped DQN, there is still value in shaping with an inferior teacher because it stabilizes the training process and will develop better policies compared to the unshaped model on average. The fact that the training is more stable may build trust for firms to use reward shaping, even if DRL remains a so-called black box method.

The bottom part of Figure 4 displays the learning curves of the best iterations of each DRL model for experiment 2 with $m = 2$, smoothed over 10 periods (other experiments revealed similar graphs). While the final performance of shaped-B and unshaped DQN is similar (see also Figure 2), we observe that the learning process of the shaped DQN is faster and more stable. Hence, even an inferior teacher policy, such as the base-stock policy, can be instrumental as the shaped model will require less computation time to obtain the same performance. Thus, computational savings

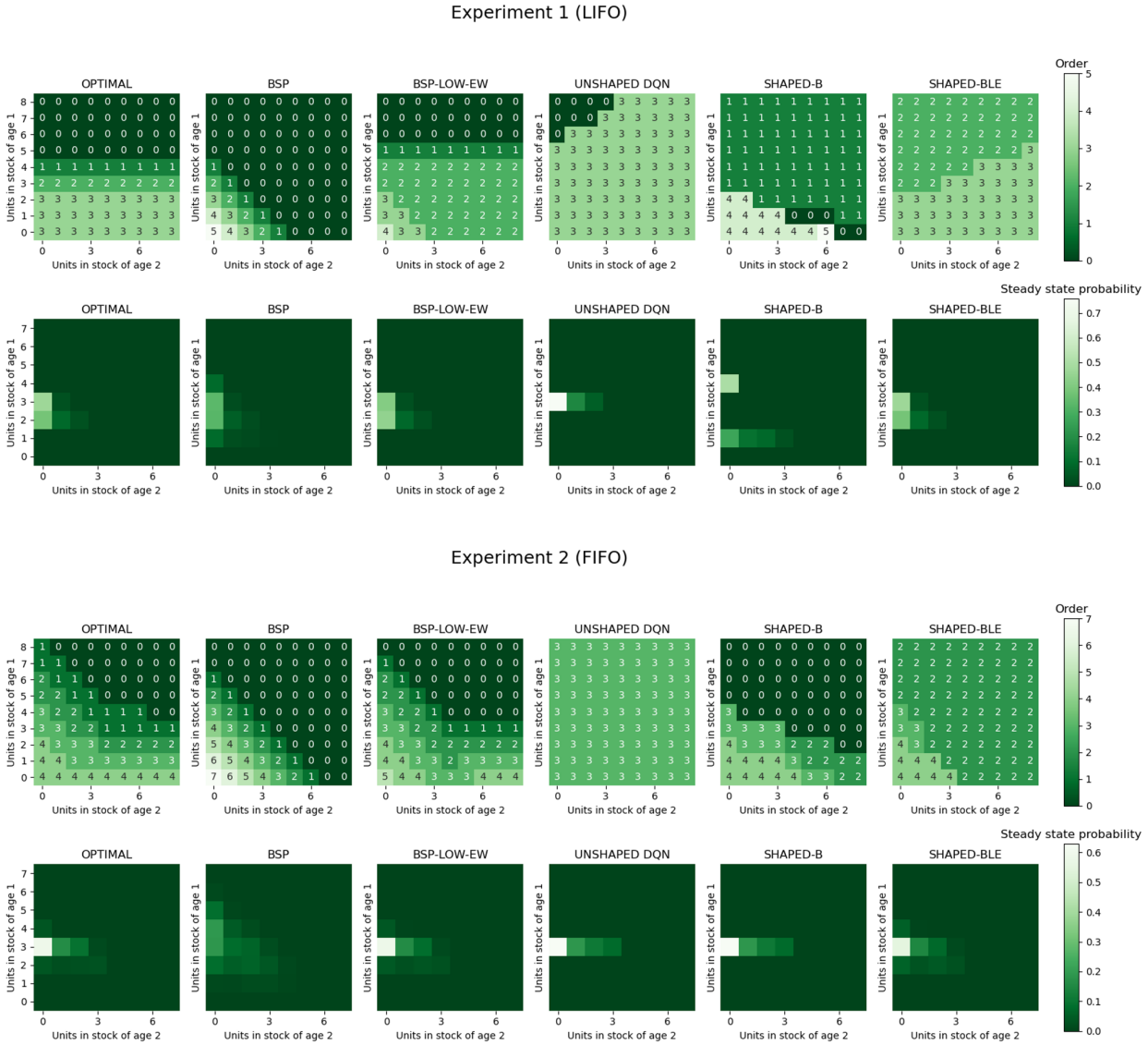

Figure 3: Visualization of different policies and steady state probabilities in experiment one and two (with $m = 2$). For each map, the x-axis represents the units in stock of age two, while the y-axis represents the units in stock of age one. The top row of each experiment represents the policy, i.e., the number in the heat map denotes the order quantity. The bottom row provides the steady state probability, i.e., the probability of being in each specific state.

do not arise because a superior policy is found, but because the same policy can be found at lower computational cost. Note, however, that the computational requirements needed to optimize the teacher policy should be taken into account. Nonetheless, in our setting the computational effort for tuning the teachers is negligible compared to the requirements for training the neural networks.

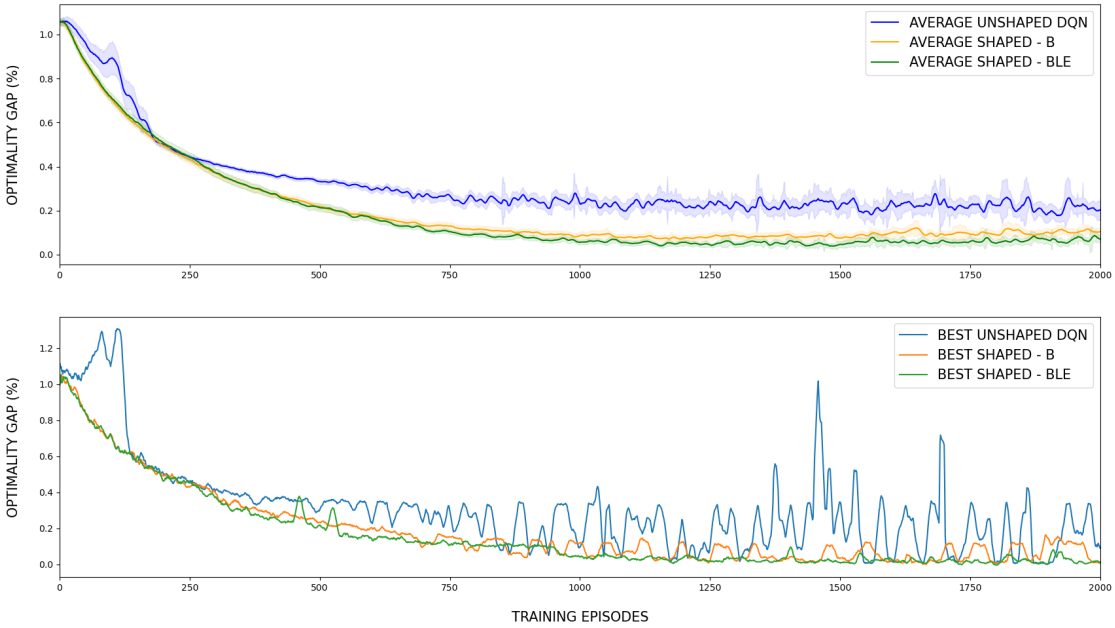

Figure 4: Optimality gap of the average (top panel) and best (bottom panel) learning curves of the different DRL models for experiment 2 with $m = 2$, smoothed over 10 periods. The shaded areas around the curves in the top part represent 95% confidence intervals.

5.3. Extending the experiment to longer lifetimes

We test the robustness of our results by extending the product lifetime to $m = \{3, 4, 5\}$, while keeping the other parameters the same. As these problems are too large to obtain the optimal policy numerically, performance is reported relative to the BSP-low-EW policy. Moreover, due to the increased dimensionality, average cost per period is calculated via simulation, rather than Markov chain analysis. Simulations were run for 25 iterations of one million time steps each, of which the first thousand are warm-up periods to eliminate the effect of the starting inventory. For the DQN algorithms, we used the same hyperparameters as those for the experiments with lifetime $m = 2$, except for the number of neurons in the input layer which are dependent on the lifetime.

Figure 5 shows that when the product lifetime m increases, the base-stock policy's relative performance ameliorates and often surpasses unshaped DQN. As a result, contrary to the experiment in which $m = 2$, shaping with the base-stock policy now has a positive effect, resulting in an improved performance of reward shaping compared to unshaped DQN, and even exceeding its teacher most of the times. Likewise, the use of the superior BSP-low-EW policy as teacher yields an improvement compared to unshaped DQN, as was the case for $m = 2$. This underpins the importance of a good teacher heuristic, and as such motivates research on inventory theory. It must be noted, however,

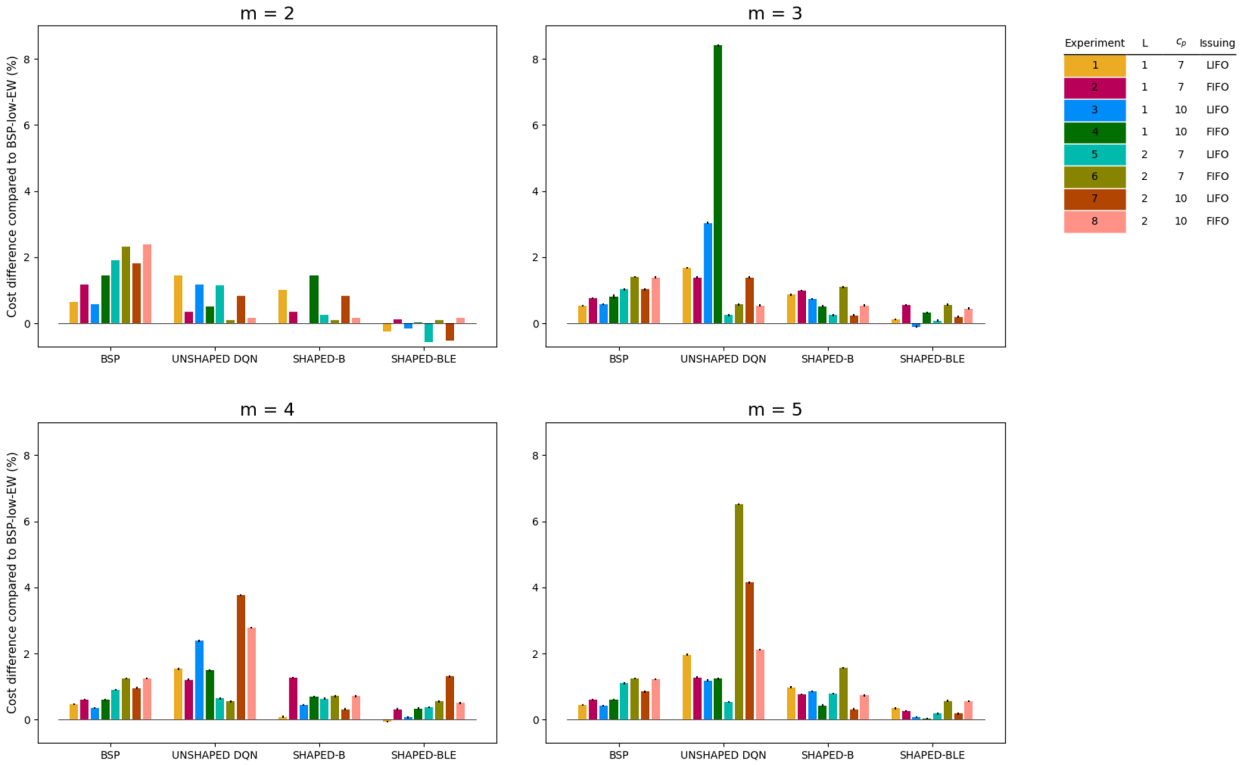

Figure 5: Relative cost difference compared to the BSP-low-EW policy of all different policies in eight settings (see Table 1) for lifetimes $m = \{2, 3, 4, 5\}$. For $m = \{3, 4, 5\}$ the long-run average cost per period is calculated via simulation, rather than Markov chain analysis. We therefore provide 99% confidence intervals for these settings.

that the shaped models seldomly outperform the BSP-low-EW policy in the experiments with $m = 3, 4$ and 5. As the base-stock policy (which is dominated by BSP-low-EW) is asymptotically optimal for increasing m for zero lead times (Bu et al., 2020), it is suspected that the optimality gaps of both heuristics decrease as m gets larger, also for the small lead times considered in these experiments. Thus, although theoretical results lack, we speculate that the performance of BSP-low-EW ameliorates for increasing m. This makes it very hard, or even impossible, for the shaped models to outperform the BSP-low-EW.

As m increases, it can be observed that the performance of unshaped DQN relative to the BSP-low-EW policy decreases considerably in some settings. We note that our results are conservative as the DRL algorithm was initially (limitedly) tuned for experiments with lifetime $m = 2$. However, with shaping, performance can be increased to a point at which it does not matter that the model was not tuned for the problem it tries to solve. These results suggest that reward shaping can reduce computationally costly hyperparameter tuning, if a good teacher heuristic is available. Furthermore,

it underlines the practical relevance for companies who want to deploy DRL models without hassle.

6. Conclusion

Potential-based reward shaping is an easy and elegant technique to manipulate the rewards of an MDP, without altering its optimal policy. We have shown how potential-based reward shaping can transfer knowledge embedded in heuristic inventory policies and improve the performance of DRL algorithms when applied to inventory management. We implemented DQN to the perishable inventory problem by shaping rewards using a base-stock and the BSP-low-EW policy as teacher policies. Whereas the former is widely used in practice, the latter is the best known heuristic to date. Our results show that reward shaping generally has a positive impact. When using potential-based reward shaping functions, we find that the average performance of DRL improves, both when using the base-stock and BSP-low-EW policy as teacher, and the training process becomes much more stable with reward shaping. When the teacher policy is superior to unshaped DQN, reward shaping generally outperforms their unshaped counterparts and occasionally also their teachers. If, however, the teacher policy performs worse than the unshaped DQN, reward shaping does not improve on their unshaped counterparts. Yet, with reward shaping the DQN will learn more per episode, meaning that it will find a better policy than unshaped DQN given a defined computation time. Thus, measuring time until a certain target performance is reached, reward shaping does reduce the total computation time (and cost), even when using an inferior teacher.

Our results show that good replenishment heuristics can improve the performance of DRL. Research on tractable inventory policies thus still remains desired, even in the (emerging) presence of general-purpose DRL algorithms. The technique presented in this paper exploits existing well-performing teacher heuristics and is simple to implement, regardless of the specific DRL algorithm and inventory problem. We believe that this may enable companies to deploy DRL models more easily, without the burden of extensive hyperparameter tuning. By open-sourcing our code[2], we hope this work can serve as a primer for future projects, hence paving the way for DRL to practical inventory applications.

[2]All code can be consulted via following Github Repository: `https://github.com/bramdemoor-BE`.

Appendix A. Hyperparameters

Table A.2 presents the hyperparameters of the DQN algorithm used in all settings for both shaped and unshaped DQN.

Hyperparameter name	Hyperparameter used
Hidden layers	2
Connection	Dense
# nodes input layer	$m + L - 1$
# nodes output layer	$q_{max} + 1$
# nodes hidden layers	32
Activation function hidden layers	ReLU
Activation function output layer	Linear
Loss	Mean Squared Error
Optimizer	Adam
Discount factor (γ)	0.99
Epsilon decay	0.997
Minimum epsilon	0.01
Learning rate	0.001
Training episodes	2,000
Batch size (N)	32
Target network update	Every 20 episodes
Replay buffer length (M)	20,000

Table A.2: Hyperparameters used when training DQN. In all settings, these hyperparameters are kept constant, both for unshaped DQN and shaped DQN.

References

Arrow, K. J., Harris, T., and Marschak, J. (1951). Optimal Inventory Policy. *Econometrica*, 19:250–272.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, USA.

Boute, R. N., Gijsbrechts, J., van Jaarsveld, W., and Vanvuchelen, N. (2021). Deep reinforcement learning for inventory control: a roadmap. *To appear in European Joural of Operational Research*.

Broekmeulen, R. A. and van Donselaar, K. H. (2009). A heuristic to manage perishable inventory with batch ordering, positive lead-times, and time-varying demand. *Computers & Operations Research*, 36(11):3013–3018.

Brys, T., Harutyunyan, A., Taylor, M. E., and Nowé, A. (2015). Policy Transfer using Reward Shaping. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, pages 181–188, Istanbul, Turkey. International Foundation for Autonomous Agents and Multiagent Systems.

Brys, T., Nowé, A., Kudenko, D., and Taylor, M. E. (2014). Combining multiple correlated reward and shaping signals by measuring confidence. In *AAAI'14: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, volume 28, pages 1687–1693, Québec City, Québec, Canada. AAAI Press.

Bu, J., Gong, X., and Chao, X. (2020). Asymptotic Optimality of Base-Stock Policies for Perishable Inventory Systems. Available at SSRN: 10.2139/ssrn.3724966.

Bulinskaya, E. V. (1964). Some Results Concerning Optimum Inventory Policies. *Theory of Probability & Its Applications*, 9(3):389–403.

Chazan, D. and Gal, S. (1977). A Markovian Model for a Perishable Product Inventory. *Management Science*, 23(5):512–521.

Chollet, F. (2017). *Deep learning with Python*. Simon and Schuster.

Cohen, M. A. (1976). Analysis of Single Critical Number Ordering Policies for Perishable Inventories. *Operations Research*, 24(4):726–741.

Cohen, M. A. and Pekelman, D. (1978). LIFO Inventory Systems. *Management Science*, 24(11):1150–1162.

Devlin, S. and Kudenko, D. (2012). Dynamic Potential-Based Reward Shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, volume 1, pages 433–440, Valencia, Spain. International Foundation for Autonomous Agents and Multiagent Systems.

Devlin, S., Kudenko, D., and Grześ, M. (2011). An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(2):251–278.

Dorigo, M. and Colombetti, M. (1994). Robot shaping: developing autonomous agents through learning. *Artificial Intelligence*, 71(2):321–370.

Duan, Q. and Liao, T. W. (2013). A new age-based replenishment policy for supply chain inventory optimization of highly perishable products. *International Journal of Production Economics*, 145(2):658–671.

Efthymiadis, K. and Kudenko, D. (2013). Using plan-based reward shaping to learn strategies in StarCraft: Broodwar. In *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, pages 1–8, Niagara Falls, ON, Canada. IEEE.

Fries, B. E. (1975). Optimal Ordering Policy for a Perishable Commodity with Fixed Lifetime. *Operations Research*, 23(1):46–61.

Gibney, E. (2017). Self-taught AI is best yet at strategy game Go. *Nature*.

Gijsbrechts, J., Boute, R. N., Van Mieghem, J. A., and Zhang, D. J. (2020). Can Deep Reinforcement Learning Improve Inventory Management? Performance and Implementation of Dual Sourcing-Mode Problems. Available at SSRN: 10.2139/ssrn.3302881.

Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., and Levine, S. (2019). Learning to Walk via Deep Reinforcement Learning. Available at arXiv: 1812.11103.

Haijema, R. (2013). A new class of stock-level dependent ordering policies for perishables with a short maximum shelf life. *International Journal of Production Economics*, 143(2):434–439.

Haijema, R. and Minner, S. (2019). Improved ordering of perishables: The value of stock-age information. *International Journal of Production Economics*, 209:316–324.

Haijema, R., van der Wal, J., and van Dijk, N. M. (2007). Blood platelet production: Optimization by dynamic programming and simulation. *Computers & Operations Research*, 34(3):760–779.

Harutyunyan, A., Devlin, S., Vrancx, P., and Nowé, A. (2015). Expressing Arbitrary Reward Functions as Potential-Based Advice. In *Proceedings of the Twenty-Ningth AAAI Conference on Artificial Intelligence*, pages 2652–2658, Austin, Texas, USA. AAAI Press.

Karlin, S. and Scarf, H. (1958). Inventory models of the arrow-harris-marschak type with time lag. In *Studies in the mathematical theory of inventory and production*, pages 155–178. Stanford University Press.

Kingma, D. and Ba, J. (2017). Adam: A Method for Stochastic Optimization. Available at arXiv: 1412.6980.

Kiran, R. B., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., and Pérez, P. (2021). Deep Reinforcement Learning for Autonomous Driving: A survey. Available at arXiv: 2002.00444.

Mataric, M. J. (1994). Reward Functions for Accelerated Learning. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 181–189, New Brunswick, NJ, USA. Elsevier.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–542.

Moolayil, J. (2019). *Learn Keras for Deep Neural Networks*. Springer, Spring Street 233, New York, NY, USA.

Nahmias, S. (1975a). A Comparison Of Alternative Approximations For Ordering Perishable Inventory. *Information Systems and Operational Research*, 13(2):175–184.

Nahmias, S. (1975b). Optimal Ordering Policies for Perishable Inventory - II. *Operations Research*, 23(4):735–749.

Nahmias, S. (2011). *Perishable Inventory Systems*. Springer, Spring Street 233, New York, NY, USA.

Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287, Bled, Slovenia. Morgan Kaufmann Publishers Inc.

Oroojlooyjadid, A., Nazari, M., Snyder, L., and Takáč, M. (2020). A Deep Q-Network for the Beer Game: A Deep Reinforcement Learning algorithm to Solve Inventory Optimization Problems. To appear in Manufacturing & Service Operations Management. Available at arXiv: 1708.05924.

Pierskalla, W. P. and Roach, C. D. (1972). Optimal Issuing Policies for Perishable Inventory. *Management Science*, 18(11):603–614.

Randløv, J. and Alstrøm, P. (1998). Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471, Madison, Wisconsin, USA. Morgan Kauffmann Publishers Inc.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. (2020). Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Silver, D., Huang, A., Maddison, C. J., 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., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362:1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.

Skinner, B. (1938). *The behavior or organisms: an experimental analysis*. D. Appleton & Company, USA.

Thompson, N. C., Greenewald, K., Lee, K., and Manso, G. F. (2020). The Computational Limits of Deep Learning. Available at arXiv: 2007.05558.

Vanvuchelen, N., Gijsbrechts, J., and Boute, R. N. (2020). Use of Proximal Policy Optimization for the Joint Replenishment Problem. *Computers in Industry*, 119:103239.

Wiewiora, E., Cottrell, G., and Charles, E. (2003). Principled Methods for Advising Reinforcement Learning Agents. In *ICML '03: Proceedings of the Twentieth International Conference on Machine Learning*, pages 729–799, Washington, DC, USA. AAAI Press.

Williams, C. L. and Patuwo, B. (1999). A perishable inventory model with positive order lead times. *European Journal of Operational Research*, 116(2):352–373.