FACULTEIT ECONOMIE
EN BEDRIJFSKUNDE

REINFORCEMENT LEARNING FOR INVENTORY OPTIMISATION IN MULTI-ECHELON SUPPLY CHAINS

Word count: 15.892

Victor Hutse
Student number : 0140756292

Supervisor: Prof. Dr. Francis wyffels

Master's Dissertation submitted to obtain the degree of:

Master in Business Engineering: Data Analytics

Academic year: 2018-2019

UNIVERSITEIT
GENT

FACULTEIT ECONOMIE
EN BEDRIJFSKUNDE

REINFORCEMENT LEARNING FOR INVENTORY OPTIMISATION IN MULTI-ECHELON SUPPLY CHAINS

Word count: 15.892

Victor Hutse
Student number : 0140756292

Supervisor: Prof. Dr. Francis wyffels

Master's Dissertation submitted to obtain the degree of:

Master in Business Engineering: Data Analytics

Academic year: 2018-2019

UNIVERSITEIT
GENT

Deze pagina is niet beschikbaar omdat ze persoonsgegevens bevat.
Universiteitsbibliotheek Gent, 2021.

This page is not available because it contains personal information.
Ghent University, Library, 2021.

Preface

Since I first came in contact with the principles of machine learning about two and a half years ago, my interest in the subject has kept growing. Partly because I am deeply fascinated by how all these methods work and partly because I believe that it can make a difference in the end. This thesis is the product of that growing fascination.

The true journey of this thesis started last summer when I had decided I would write something about reinforcement learning. At this point, I could fit my knowledge of reinforcement learning into a handful of sentences. Over summer I started by following David Silver's course on Youtube, rigorously taking notes. Because I was determined to keep learning on the subject I ended up bringing a bundle of about 300 pages worth of RL papers with me while trekking in the Himalayas, reading them early before breakfast and late before going to bed.

I would like to extend special thanks to Prof. Dr. Francis wyffels and Andreas Verleysen. I am very grateful for the guidance you gave me, the countless practical tips on how to realise the RL program, and the constant availability. I truly enjoyed the moments where we could brainstorm and share thoughts, during the meetings that always seemed to run over.

Next, I would like to thank my family and friends who continually supported me. They were prepared to listen when I was stuck, when something would not work, and when something finally did work.

Looking back over the past year I am glad that I chose this subject. I can honestly say it never lost my interest. I hope I may learn as much in the coming year as I have in the past year.

The corresponding code base will be made available at
`https://github.ugent.be/vhutse/gym-scm`.
The icons used in schematic figures were taken from Flaticon.com.

Victor Hutse, Augustus 2019

To Raymond Hutse

Thank you for always believing in me.

Reinforcement learning for Inventory Optimization in Multi-Echelon Supply Chains

by

Victor HUTSE

Abstract

This thesis is inspired by the recent success of reinforcement learning applications such as the DQN Atari case, AlphaGo and more specific uses of reinforcement learning in the Supply Chain Management domain. We build on the paper by Kemmer et al., where reinforcement learning is used for general inventory optimisation in a double echelon supply chain. We extend the existing literature in two ways. First by increasing the problem complexity: we add difficulty through non-zero lead times, continuous action spaces and by regarding larger systems. Secondly, we use more recent reinforcement learning algorithms like DQN and DDPG using function approximation, which become necessary for more complex problems. During the process, we have built a simulation environment conform to the OpenAI Gym API to facilitate future benchmarking in this environment.

Keywords

Reinforcement learning, Supply chain optimisation, Inventory optimisation, Machine Learning, Multi-echelon, Operations management

Contents

1 Introduction **1**
1.1 Problem Description 1
1.2 Current solutions 3
1.3 Research Question 3

2 Inventory Optimisation **6**
2.1 Theory of inventory optimisation 6
2.1.1 Setting the lot size: Economic Order Quantity 7
2.1.2 Extensions of the EOQ model 8
2.1.3 Setting the reorder point 8
2.2 Current Inventory Management methods 9
2.2.1 Heuristic policy: (Q,R) 9
2.2.2 Multi-echelon methods 11

3 Reinforcement Learning **12**
3.1 Reinforcement Learning methods 12
3.1.1 Q-learning 13
3.1.2 Neural fitted Q-iteration and Deep Q learning 14
3.1.3 Deep Deterministic Policy Gradient 16
3.2 Inventory Optimisation from a RL perspective 18
3.2.1 The state space 19
3.2.2 The action space 21
3.2.3 The reward function 22
3.3 Improving Inventory Optimisation with Reinforcement Learning . . . 24
3.3.1 The choice of the lot size 24

3.3.2 Determining the reorder point 24

4 Experiments with Discrete Action Spaces 26

4.1 Base case: Discrete approach 27

4.1.1 Experiment 27

4.1.2 Results and Discussion 32

4.2 Expanded case: Discrete approach 36

4.2.1 Experiment 36

4.2.2 Results and Discussion 38

5 Experiments with Continuous Action Spaces 40

5.1 Experiment 40

5.1.1 Action and state space 42

5.1.2 Demand function 42

5.1.3 (Hyper)parameter setting and setup 42

5.2 Results and discussion 49

6 Conclusion 54

A Complete results tables 61

List of Symbols

Due to the crossing of several research fields, this work requires a large amount of symbols to be used in the mathematical functions. The double usage of symbols has been avoided to the extent possible, in the remaining cases the intent should be clear from context. The variables are grouped by use. The first block is used for the specification of the environment, the second is used in the context of RL and the third is used in the domain of SCM.

$I_{l,p}$	Inventory position per location and per product
$D_{l,p}$	Demand per location and per product
$D_{l,p}^{\text{avg}}$	Average demand per product and per location
$P_{l,p}$	Production action per product and per factory
$S_{f,s,p}$	Transportation action per factory, shop and product combination
ρ_p	Price per product
$\Pi_{l,p}$	Production cost
$H_{l,p}$	Holding cost
$T_{l,p}$	Transportation cost
$K_{l,p}$	Setup cost
$\Omega_{l,p}$	Opportunity cost of stock-out
$\pi_{f,p}$	Unit production cost
$\eta_{l,p}$	Unit holding cost per location and per product
C_p	Truck capacity per product
τ_l	Truck cost
$\kappa_{f,p}^{\text{T}}$	Transportation setup cost per factory and per product
$\kappa_{f,p}^{\text{P}}$	Production setup cost per factory and per product
$\sigma_{l,p}$	Stock-out penalty per product
t	Time step

V	Variance change
M	Trend
$\epsilon_{s,p}$	Demand perturbation per product and shop
Q^*	EOQ amount
λ	Demand rate
c	Cost per item ordered or produced
R	Reorder point
L	Lead time
s	Safety stock
h	Holding cost per unit per unit time
K	Setup cost
$L(z)$	Normalised loss function
$n(R)$	Expected number of stock-outs in the lead time for the (Q,R)-model
p	stock-out cost
Q	Order amount
R	Reorder point
$G(Q,R)$	Expected average cost for the (Q,R)-model
μ	Mean demand
σ	Standard deviation of the demand
Q	Q-value or Q-network
μ	Actor network
R	Reward
S	State
A	Action
γ	Discount rate
θ	Weights of a neural network
P	State change probabilities
I	Inventory state
W	Production and Transportation state
D	Demand state

List of Abbreviations

DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q Networks
EOQ	Economic Order Quantity
Hold.	Holding costs
Inv.	Inventory
MDP	Markov Decision Process
NFQ	Neural-Fitted Q iteration
Opp.	Opportunity cost of shortage
Prod.	Product, production or production costs
RL	Reinforcement Learning
SCM	Supply Chain Management
Trans.	Transportation or transportation costs

Chapter 1

Introduction

With the use of Reinforcement Learning to achieve inventory optimisation we find ourselves at the cross-section of the fields of business and computer science. The problem we are studying situates itself in the field of business. The overall goal is to successfully run a business. To achieve this, amongst other things we attempt to minimise the costs made concerning making, ordering and holding products, while safeguarding the revenue.

While this might seem like a trivial problem for small companies, the number of products companies need to hold in inventory quickly increases for larger companies. The difficulty of this problem increases when we take supply chains as a whole into account, either all belonging to one company or consisting of multiple companies. The inventory must then be managed for a vast amount of products over different locations, including dependencies across products and locations. All this renders it difficult to keep an overview over all inventory levels, even using the computer software tools available today.

Bearing this in mind and inspired by the recent successes of applications of Reinforcement Learning in other domains we attempt to make an RL agent to manage the inventory levels of a supply chain.

1.1 Problem Description

To investigate the abilities of an RL agent in this domain we take the case of a small supply chain, producing and selling a certain product. The supply chain consists of one factory and three shops. The product is produced in the factory and when necessary transported to the shops from where they can be sold according to the external demand from customers. We

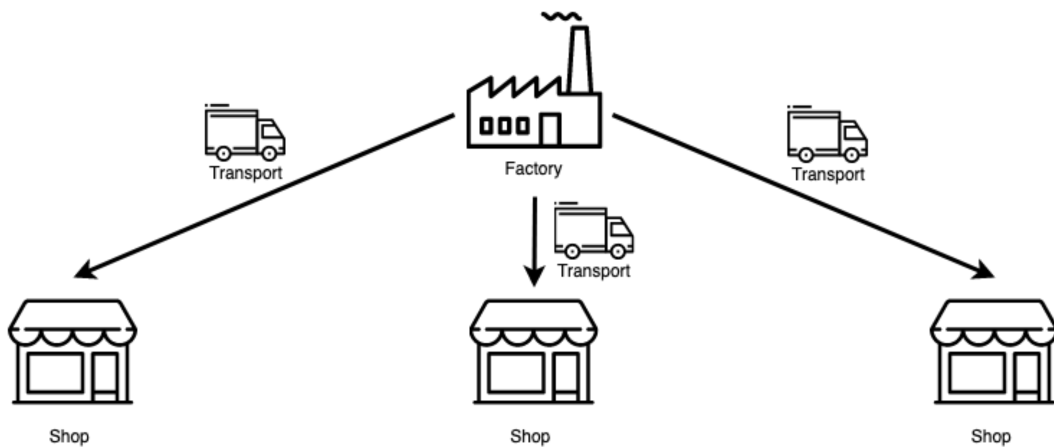


Figure 1.1: Base case: One factory, three shops and one product. This is the the structure of the supply chain used as a red line through this work.

assume that the necessities for the production process are always available. Furthermore, we assume that customers do not backorder products when they are not available. In such a case, customers go to another shop instead.

The goal of an agent in this environment is to maximise the profit across the *whole* supply chain, achieved over a certain period. We define the profit as the attained revenue minus the costs incurred due to production, holding products, transportation of products, setup and stock-outs. The reward function is mathematically specified in section 3.2.3.

Because the algorithms behind RL agents have sprung from the idea of learning from experience [30], an extensive volume of data is required to be able to learn from. To this end we have created a discrete event simulation model, to be able to create this data artificially. This placed us in control of the data supply and makes it possible to generate interesting scenarios. This allows the agent to be tested and developed in 'laboratory' circumstances, before exposing it to the complexity of real-world data. The environment simulation model was made according to the framework of the OpenAI Gym [6].

The main complexities added to the environment are the non-zero lead times, different demand scenarios, production constraints and inventory constraints. An environment is characterised by positive lead times when the products are not immediately available when produced or transported. They become available after several time steps. The production constraint put a limit on the number of products that can be set for production in every time step. The inventory constraints limit the number of products that can be held at a location.

The different demand scenarios are all based on a sinusoidal demand function, where some parameters can be changed and tuned in or out. Inspiration for this artificial demand signal was found in Kemmer et al. [18]. The demand is made stochastic by adding a stochastic signal to the demand that perturbs the sinusoidal form. It is possible to introduce a trend in the demand function or make it heteroskedastic. The demand function is mathematically introduced in section 4.2.1.

1.2 Current solutions

The current solutions to the problem of inventory optimisation are all rather similar. In effect, they boil down to the creation of a simple heuristic, often characterised by two variables. Three of the most well-known examples are the (Q,R)-policy, the (s,S)-policy and the (S,T)-policy [35]. The (Q,R)-method specifies a fixed order quantity and a reorder point in terms of inventory level. The (s,S)-method specifies a reorder point s and a level S to order up to, both in terms of inventory level. The (S,T)-method indicates a point to order up to S, in terms of inventory level. The variable T indicates a fixed time interval for when to order. These policies are all specified upfront and updated periodically based on historic demand.

The methods used to determine a good value for the order size and the reorder point range from the store owners personal experience to rather complex mathematical and statistical models devised to minimise the inventory costs over different locations. The order quantity is usually based on the Economic Order Quantity theory or one of it's more recent extensions. While the reorder point is most often set to an amount that allows the company in question to attain a certain service level, assuming some statistical properties of the demand function.

1.3 Research Question

The main research question we attempt to answer in this work is:

Q1: Can current Reinforcement Learning methods compete with incumbent inventory optimisation methods?

With '*current Reinforcement Learning methods*' we refer to recent well-known reinforcement learning methods that have been used, tested and implemented by other authors. Without adding any special flavours. The inventory optimisation methods we compare RL methods with

are simple heuristic policies such as the (Q, R) policy. We opt to use this policy as a baseline because it is simple to implement and it well represents the general principles of current inventory optimisation methods. There are more complex methods available today, but we believe these are more popular in academic circles while gaining less traction in practice. Compared to the (Q, R) heuristic we define '*being able to compete with*' as performing better than the heuristic method. First of all, because this heuristic is only the very basis of inventory optimisation. Secondly, because using an RL agent to manage the inventory would probably feel like giving up a part of the control to most people. We assume that if the agent is not able to outperform the incumbent methods, a user would not deem it worthwhile to give up this part of the control.

We devise a set of sub-questions to further determine in which aspects RL agents can compete with incumbent methods, to further elaborate our comprehension of the capabilities of RL based methods in this domain. The first sub-question is:

Q1.1: Can reinforcement learning provide a more versatile approach for inventory optimisation?

With this research question, we wish to investigate the possibility for RL methods to learn to recognise and exploit patterns in the external demand and the flow of inventory through the supply chain. We believe an RL method could exploit this understanding by dynamically changing the used order size and the de facto reorder point. While incumbent methods appear to be more static.

Q1.2: Can reinforcement learning provide a more integrated approach to inventory optimisation in supply chains?

With this research question, we intend to examine whether RL methods can identify more efficient policies with respect to how many products of every type to hold at every location and with respect to order amounts. Can profit be increased by synchronising orders for different product categories etc. The final research question is more practical and more inherent to the reinforcement learning domain:

Q1.3: Are current reinforcement learning methods scalable in approaches to inventory optimisation in larger supply chains?

This last sub-question to the first research question is a crucial point for the use of RL in inventory management. The scalability in terms of problem size for RL approaches to inventory

management will probably prove decisive with respect to the future success of these methods in the domain of inventory management. While the problems regarded in this study might seem trivial at first, they are considerably large compared to other RL applications.

Q2: Are continuous action space RL algorithms competitive when the problem becomes too large for discrete action spaces?

Problems with a limited action space are generally more cost-efficiently solved using a discrete action space. However, this approach is not scalable to problems with larger action spaces. This is due to the fact that every action is regarded independently, making it impossible to take advantage of the similarity between actions. We research whether continuous action spaces are competitive for the considered case.

Chapter 2

Inventory Optimisation

This thesis deals with the topic of inventory optimisation. The essence of inventory management is the art of balancing on the line between an excess and a shortage of inventory. Both holding too much and too little inventory can prove problematic for the functioning and profit of a company. The difficulty of this exercise lays with the problem of unknown future demands.

Different approaches have been used to address the problem of inventory management. In less scientific situations the policy is undoubtedly hinged upon the experience of the shop owner or responsible employee. The notion of past demand evolution helps to improve current decision-making abilities. Another method is to study the past demand for each product and determine its statistical properties. This allows calculating the optimal inventory policy under the assumption that the derived statistical properties will be true in the future. More recently this last approach has been extended by combining the statistical properties of demand at different locations, to determine an optimal policy for a larger system.

This chapter starts with an introduction to the principles of the inventory management research field. Next, we give an example of how this can be used in practice. Followed by the description of the benchmark method used in this study. Finally, we give an overview of what we believe to be the strengths and weaknesses of current inventory management practices.

2.1 Theory of inventory optimisation

For the most part of the recent evolution of inventory management revenue was not considered, i.e. revenue loss caused by shortages is assumed not to happen [29]. The optimal inventory policy was determined by minimising the costs tied to inventory management. The most well-

known example of this theory is the Economic Order Quantity (EOQ), first introduced by F.W. Harris [11]. While Harris' intended use for his findings was to determine the optimal production lot sizes, his paper eventually largely shaped the field of inventory optimisation.

2.1.1 Setting the lot size: Economic Order Quantity

The basic EOQ model makes the following assumptions:

1. The demand is constant and known. It is represented by the demand rate of λ.
2. Stockouts are not allowed.
3. Orders are completed immediately. Which means the lead times are zero.
4. The costs taken into account are:
   (a) A fixed setup cost K per order that is made.
   (b) A cost per item ordered or produced, c
   (c) The cost to hold a unit per time unit, h.

Because stockouts are not allowed by the assumptions of the EOQ model, the revenue is not influenced by the decisions it makes. This is why optimal results are achieved by minimising the costs, which is equivalent to maximising the profit in the case of constant revenue. The to be minimised cost function of the EOQ model is:

$$T(Q) = \frac{Q}{2}hc + \frac{\lambda}{Q}K + c\lambda. \tag{2.1}$$

Q is the order size, this is a fixed amount. As the lead times are assumed to be zero, it can easily be seen that the optimal strategy is to only place a new order when we have run out of inventory. The costs are minimised by ordering in batches of Q^*. With Q^* equal to:

$$Q^* = \sqrt{\frac{2K\lambda}{h}} \tag{2.2}$$

These very simple principles have been the basis of the conceptual framework for inventory optimisation in the past 100 years [7]. Step by step, authors have contributed by relaxing assumptions and including additional complexities. This has lead to a series of rather complex mathematical models that are each applicable to a very narrow set of cases with strict assumptions.

2.1.2 Extensions of the EOQ model

While the basic EOQ model and it's direct extensions such as the adaption for non-zero lead times are simple and easy to grasp, the more advanced models are complex and difficult to implement. A different model needs to be developed for every scenario [1, 28].

2.1.3 Setting the reorder point

In section 2.1.1 we mentioned that under the assumption of zero lead times and known demand the optimal strategy is to place a new order when the inventory level drops to zero. When relaxing these assumptions, an order should be placed before the inventory depletes: the inventory level at which to place an order is known as the *reorder point*. The reorder point should include the amount of demand expected to occur during the lead time of the order, supplemented by an amount to account for the uncertainty of the demand. The latter is referred to as the safety stock. This results in

$$R = \lambda L + s, \tag{2.3}$$

where λ represents the demand rate, L the lead time and s is the safety stock.
The reorder point is dependent on a series of system characteristics: the demand uncertainty, the lead time, the cost of holding inventory, and the cost of not being able to fulfil demand. In practice, two approaches are used to determine the reorder point based on these factors. The first is the *trade-off method*, in which balance is sought between the cost of holding inventory and the cost of unfulfilled demand. Because the latter is not directly observable, it can be difficult to quantify. This leads to the second approach, the *service level method*. In this case, the reorder point is chosen to attain a certain service level, given the lead time and the demand uncertainty.

Here we have defined the cost of unfulfilled demand because it is part of the reward function of the RL Agent. Thus we focus on the trade-off method to let both agents be based on the same information. As both methods require the introduction of a considerable amount of equations we limit the discussion to the trade-off method. The service level method is not used in this work.

The trade-off method

When the cost of unfulfilled demand is defined, the trade-off method can be used. In this case the cost function can be defined as the average cost of holding, setup, and unfulfilled demand. As per Nahmias and Olsen (2015, p.267) [23], this results in:

$$G(Q,R) = h(Q/2 + R - \lambda L) + K\lambda/Q + p\lambda n(R)/Q. \quad (2.4)$$

The terms in the right-hand side of this equation represent the holding cost, setup cost and shortage cost respectively. The factor $n(R)$ represents the expected number of shortages. The minimisation of this cost function is achieved by iteratively solving the following two equations:

$$Q = \sqrt{\frac{2\lambda[K + pn(R)]}{h}} \quad (2.5)$$

$$1 - F(R) = Qh/p\lambda. \quad (2.6)$$

For a normally distributed demand, $n(R)$ can be computed with the standardised loss function, which is defined as

$$L(z) = \int_z^\infty (t - z)\phi(t)dt. \quad (2.7)$$

Following which, it has been found that

$$n(R) = \sigma L\Big(\frac{R-\mu}{\sigma}\Big) = \sigma L(z). \quad (2.8)$$

In equation (2.6), $F(R)$ refers to the value found for the cumulative function of the normally distributed demand. The value found for $1 - F(R)$ corresponds with a z value. Which can be used to calculate the reorder point R as $R = \sigma z + \mu$.

2.2 Current Inventory Management methods

2.2.1 Heuristic policy: (Q,R)

The baseline method discussed here is the (Q, R) inventory policy. This method assumes the inventory position to be monitored continuously. When the inventory position drops to the reorder point R, a new order of Q units is placed.

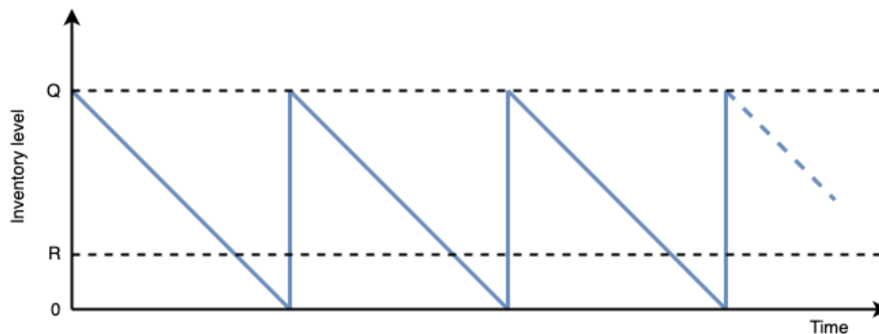


Figure 2.1: (Q,R)-method inventory level pattern. This is the characteristic pattern of the inventory level for heuristic methods. An order is placed when the inventory level drops to the reorder point R. Products are ordered in fixed batches of Q units. When the inventory level reaches R again after the replenishment, the cycle repeats itself.

Thus to define a working heuristic method, two parameters need to be defined: the lot size Q and the reorder point R. In section 2.1.3 one method to do this is presented. This method is developed for systems with stationary, normally distributed demand. Characterised by positive lead time, setup costs, holding costs, proportional ordering costs and a shortage cost. This method starts by setting the lot size Q to the optimal amount defined according to the EOQ theory, Q^*. Followed by iteratively solving equation (2.5) and (2.6), until the values for Q and R each converge upon a value.

As the condition of normality is not satisfied in the demand function used in the environment at hand, we use a slightly adapted method here. For the lot size, we use the Q^* found by the EOQ theory. We define R by filling in equation (2.6) once. We do not perform the iteration of both equations to mitigate the influence of the normality assumption.

Advantages

The main advantage of the EOQ-based heuristic methods is that they are very easy to implement. Every company can make at least a basic estimate of the costs related to ordering and keeping goods, the lead time and the probability distribution of the demand. Together with the desired service level, such an exercise will yield valuable insights into the near-optimal lot size and reorder point for a location. Finally, it is easy to be set up in a very decentralised manner.

Disadvantages

While getting an approximate solution to small problems is very easy with EOQ-based heuristic methods, achieving good results quickly becomes more complex as the problem becomes more difficult. On the one hand, defining the optimal lot size becomes more difficult because the extended versions of the EOQ model grow very mathematically complex as the assumptions are made more realistic. This leads to a large range of complex mathematical models that each apply to very specific situations and assumptions. On the other hand, the quality of the reorder points determined by the heuristic methods deteriorates. The practice of managing every product and location separately leaves more and more room for improvement as the number of products and locations increase. Finally, the heuristics are a static method: unless everything is recalculated, the practice remains the same. The increased levels of connectivity and the increased ability to store, process and analyse data is not exploited.

2.2.2 Multi-echelon methods

In multi-echelon approaches, the aim is to increase the efficiency of a supply chain by managing its different components jointly. For example the case of multi-echelon safety stock optimisation [9, 10]. In such cases, the lot sizes of products remain the same: defined by the EOQ model or one of its extensions. The efficiency of the supply chain is instead achieved by shifting the levels of safety stock while retaining the service level offered to the final external customer.

Advantages

These kinds of methods offer an improvement for the second issue listed for EOQ-based heuristic methods. The multi-echelon methods no longer regard every location separately. Rather than taking a service level goal into account for every location, the supply chain is viewed as a whole. Only the service level to the external customers is taken into account. As can be seen in Desmet et al. (2010), the service level to external customers can be retained with less safety stock by keeping less stock in intermediate steps and more in the last level. Leading to serious cost reductions.

Chapter 3

Reinforcement Learning

Reinforcement learning is a paradigm of machine learning where an agent learns a certain behaviour, learns to link actions to situations. In supervised learning, the agent learns by comparing input data and the provided label for this case. In reinforcement learning, this labelled data is not available. Instead, the agent learns through interaction with the environment as presented in figure 3.1.

The goal of the reinforcement learning agent is to adapt its policy to maximise the reward signal it receives from the environment. The policy of an RL agent is the mechanism that determines which action to take given the current state of the environment. In the next sections, we will see that the policy can take multiple forms.

Reinforcement learning problems are specified as Markov Decision Processes. There are three parts to an MDP. The first is the agent, the entity that learns. The second is the environment, this comprises everything else and is not under control of the agent. Finally, there is the reward: a numerical feedback signal the agent receives from the reward. The goal of the agent is to maximise this reward signal through the choice of its actions.

3.1 Reinforcement Learning methods

There are different types of reinforcement learning algorithms such as model-based, value learning and policy gradient algorithms. For a detailed introduction to the field of reinforcement learning, we refer the interested reader to the base work by Sutton and Barto [30, 31].
In this document, we use two different algorithms as a basis for the learning agent. The first is Neural-fitted Q iteration [25]. This algorithm is the precursor to the Deep Q Networks algorithm

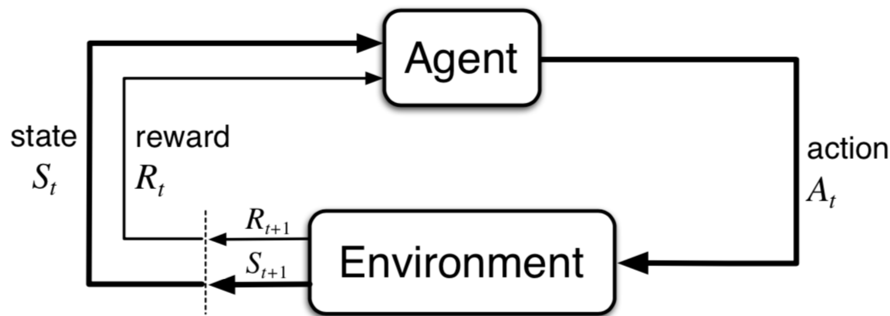


Figure 3.1: The Agent-Environment interaction in an MDP [31]. The agent interacts with the environment by selecting an action, based on the perceived state of the environment. Base on the agent's action, the state of the environment changes. The new state is in turn perceived by the agent. The environment also passes a reward signal, which is used by the environment as a feedback signal, during training.

used to play Atari [22], but using a multi-layer perceptron instead of a deep convolutional neural network, because we are not working with image data. The second algorithm is Deep Deterministic Policy Gradient (DDPG) [20]. While NFQ works with a limited set of discrete actions, DDPG allows the agent to choose its actions from a continuous action space. In the remainder of this introduction to reinforcement learning, we will describe the workings of these two algorithms. The two algorithms have Q-learning as a shared basis, so we start with a discussion of Q-learning and build up from there.

3.1.1 Q-learning

Q-learning [8] is an off-policy, model-free, value-based reinforcement learning algorithm. Off-policy means that the agent learns about using one policy while using another. In practice, this means that the agent learns about using a greedy policy, which means always choosing the action leading to the best outcome while using an ϵ-greedy policy, which means choosing a random action in $1 - \epsilon$ of all cases, to explore the state space. Model-free means that the transitions of the environment are not known, the agent only perceives the environments states. Value-based means that the agent learns a value representation of the environment, based on which a policy is handled. In the case of Q-learning, the agent learns action-values Q(s,a), which correspond to the sum of the discounted expected future rewards for an agent following a certain policy. The state-action value $Q(s, a)$ in a certain time step is recursively defined by an equation

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 3.2: The Q-learning algorithm as described by Sutton and Barto [31]

known as Bellman equation.:

$$Q(s_t, a_t) = R(s_t, a_t) + \max_{a'}(\gamma Q(s_{t+1}, a'). \tag{3.1}$$

After each iteration the action value for the corresponding state-action pair is updated in the following manner:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R + \max_{a'}(Q(s_t, a')) - Q(s_t, a_t)]. \tag{3.2}$$

This update rule can be described as updating the current Q-value in the direction of the target Q-value. This is the Q-value found by using equation (3.1) on the perceived transition. This concludes the algorithm represented in figure 3.2.

This type of approach is called a table look-up method because the current Q-value for each state-action pair is stored in a table separately. This type of method works well for problems with a relatively small state and action space but becomes exceedingly slow when these grow larger. The limitation of Q-learning is intuitively clear from the fact that in a table-based method all state-action pairs must be visited individually for their value representation to be updated. The fact that the value representation is defined by a recursive relationship in Q-learning amplifies the limitation, because every state-action pair must be updated, and thus visited multiple times to achieve a good estimate.

3.1.2 Neural fitted Q-iteration and Deep Q learning

To deal with larger state and action spaces, we can use function approximators such as neural networks to store Q-values more efficiently. This can only be done because we expect similar

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 3.3: The DQN algorithm [22]

state-action pairs, i.e. state-action pairs that are in each others vicinity, to have similar Q-values. This is what is done in more advanced reinforcement learning algorithms such as Deep Q Networks and Neural-fitted Q iteration.

Strictly speaking, the Q-learning algorithm could be used with a neural network as a function approximator without alteration. However, when we update the neural network representing the action values, we cannot alter one single value as in the table-based case. Instead, we adapt the weights of the neural network in the direction of steepest descent. This weight adaption might cause unrelated other parts of the network to be affected as well. This effect can be seen as an advantage, but in the paper on NFQ [25] Riedmiller suggests that this most often leads to unstable and slow training.

To remedy this problem methods such as NFQ and DQN use a technique called experience replay [21]. This practice involves storing the sequences in the form of (s,a,s',r), that the agent has gone through. The network is then updated using stochastic gradient descent [5] on a random sample from experience buffer. This involves calculating the gradient for a loss function based on the differences between the current Q-values and the targets, similar to equation (3.2). The network is then updated by pulling the calculated gradient through the network using backpropagation [13, 26]. While normal reinforcement learning data is highly correlated, this practice of experience replay allows us to train on batches of uncorrelated samples more closely

representing the whole state-action space during every step. This results in the agents faster and more reliable convergence. An overview of the resulting algorithm is found in figure 3.3.

Deep Q learning has been improved by several sources since its first introduction. Some of these improvements are used in this study. Van Hasselt, Guez, and Silver [34] add a target network to the DQN algorithm. The result is called Double Deep Q Networks or DDQN. This is a second neural network with a duplicate of the weights of the Q-network. The target network is used to calculate the target values, used during the stochastic gradient descent. When the same network is used for target calculation the target value directly influences its own change, which can lead to feedback loops. Using a separate network for the calculation of the target values decorrelates the relationship between the target values and the update Q-network, stabilising the training of the network. The target network is then periodically updated using the original Q-network. DDQN uses a hard update, where the weights of the original Q-network are directly copied into the target network. Another option is the soft update, where the weights of the original network are gradually brought over to the target network. As shown in equation 3.3, with $\theta^{Q'}$ and θ^{Q} the weights of the target network and the original network respectively, and $\tau \ll 1$ the positive update coefficient:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}. \tag{3.3}$$

Schaul et al. [27] use a different method to select the transition sequences used for the gradient descent. Instead of randomly selecting transition sequences they attribute each transition sequence with a priority. The priority is calculated based on how much was learned from a transition. The authors use the intuition: "How surprised the agent was with a transition" [27]. This is represented by the absolute difference between the Q-value and the target value for a transition. The resulting algorithm is referred to as prioritised experience replay and is characterised by two hyperparameters: α and β. The α is a parameter for the degree of prioritisation. The parameter β is annealed from its starting value to one, this value corrects the bias introduced by prioritised experience replay.

3.1.3 Deep Deterministic Policy Gradient

In the methods discussed above the number of actions an agent could choose from was limited. We call this a discretised action space. DDPG is an algorithm made for continuous action

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 3.4: The DDPG algorithm [20].

spaces. In NFQ, and Q-learning, in general, a value representation was learned for all state-action combinations. In NFQ the difficulty of a large number of values to store was overcome by using a function approximator such as a neural network to store all the Q-values. When the preferred action was to be selected for the current state, the Q-value could be calculated for all actions the agent could take in this state. After this, the action with the largest Q-value could be chosen. When dealing with a continuous instead of a discretised action space, this mechanism to choose the preferred action becomes too computationally demanding because of the large number of actions to choose from. This is why a new, second function approximator is introduced: the policy network or actor.

DDPG is an actor-critic method, these methods are a hybrid combination of the policy gradient approach and the value representation approach. The actor network or policy network returns the best action given a state. The Q-network or critic network is very similar to the network used NFQ or DQN. Given the state and the action, it returns a certain value for this combination.

An overview of the DDPG algorithm can be found in figure 3.4. The algorithm starts with the initialisation of 4 networks: a critic network, an actor network, and a target network for each

of these. The target networks were created to introduce more stability in the training of the networks. DDPG also uses a replay buffer, which means it is also an off-policy RL method. In the first part of the main loop: when interacting with the environment, actions are selected by the policy network based on the current state. During training, a noise signal is added to the action selected by the policy network, rather than selecting a completely random action in some of the cases as done in the previously discussed methods for discrete action spaces. The noise signal is generated by a random process. The original DDPG paper suggests using noise generated by an Ornstein-Uhlenbeck process [33]. The noise from the Ornstein-Uhlenbeck process is recommended because auto-correlated noise is said to lead to more efficient exploration. As in NFQ and DQN, DDPG stores the transition sequences in a replay buffer. In the second part of the main loop, a batch of transitions is randomly sampled from the replay buffer. For this batch of transitions, the target Q-values are calculated using parameters of the target actor and target critic network. The critic is updated using these targets just as the Q-network is updated in NFQ and DQN. The actor network is updated using gradient ascent over this batch, to maximise the Q-value or expected future rewards, of the action selected for every state. The critic network is often regularised using L2 regularisation [20, 14]. When using L2 regularisation a fraction of the sum of the squared Euclidean norms of the weight matrix of each of the network's layers is added to the loss function. This introduces a bias towards smaller weights and is used to prevent overfitting. Finally, the target networks are slowly updated from the actor and critic network respectively, using soft updates as in equation (3.3).

3.2 Inventory Optimisation from a RL perspective

To make decisions in the supply chain management domain using RL, the problem at hand must be specified explicitly in mathematical terms. As mentioned the underlying mathematical principles of RL are based on MDPs. Thus we must define the problem of inventory optimisation as an MDP, which is characterised as follows: $(S, A, P_\alpha, R_\alpha)$. Where S is a set of states, A is a set of actions, P_α is a set of probabilities that taking a certain action α in a certain state will lead to another specified state. R_α is the immediate reward following the transition from one state to another. The RL methods used in this study are model-free RL methods. Such methods consider the transition probabilities to belong to the domain of the environment. Thus we do not need to explicitly satisfy the transition probabilities.

The body of literature on this subject is very limited, which is why we suggest a possible specification of the MDP. The model is an extended version of the model used by Kemmer et al. [18].

3.2.1 The state space

The state of an MDP should represent the situation of the environment. The state should be as close as possible to possessing the Markov property [30]. This means that the transition probabilities only depend on the current state and the chosen action. This can be interpreted as the current state containing all necessary information to determine the next action. In such a case the estimate of the best next action can not be improved by adding more information to the state.
It is intuitively clear that attaining this property is not possible for most realistic situations. This is certainly the case for inventory management. Inventory management inherently includes a forecasting problem, making the achievement of the Markov property impossible. We attempt to make the state as exhaustive as realistically possible.

The state space we use in this document is the following:

- The inventory level per product–location combination.
- The number of products in production and the number of products in transit, per product–location combination.
- The demand levels per product–location combination, for the past X periods.

The inventory level per location is the most elementary piece of information in the inventory management system. This corresponds to the number of products available in the shops to be sold to the end customers and the number of products available in the factories to be shipped, to replenish the inventory in the shops. The inventory level per location can be represented as a matrix, keeping the inventory level per product–location combination: $I_{l,p}$ is the inventory level at location l of product p. We make the convention of using the first columns for the factories. For example with one factory, three shops and two products the inventory per location would look like this:

$$I = \begin{bmatrix} I_{0,0} & I_{1,0} & I_{2,0} & I_{3,0} \\ I_{0,1} & I_{1,1} & I_{2,1} & I_{3,1} \end{bmatrix}. \tag{3.4}$$

The number of products in production and the number of products in transit is a necessary element of the state in inventory management problems with positive lead times. We have chosen to keep one number for each combination of product and location. This results in a matrix with the same shape as the inventory per location. The production and transit values for the factories represent the work in process levels per product per factory. The production and transit values for the shops represent the number of products currently being transported *to* this shop. We do not include information on from which factory the transported products are being shipped, because this is of little importance to decisions regarding maintaining an inventory level. Information on which amount of products was shipped when or the expected time of arrival is not included. Including this kind of information would lead to a more sparse state space, which may complicate the learning process. At the same time, we believe the explicit inclusion of this information would contribute rather little to an agents performance. The first column represents the factory, thus production. The remaining columns represent products in transportation. In our example with one factory, three shops and two products, the products in production and transit would look as follows:

$$W = \begin{bmatrix} W_{0,0} & W_{1,0} & W_{2,0} & W_{3,0} \\ W_{0,1} & W_{1,1} & W_{2,1} & W_{3,1} \end{bmatrix}. \tag{3.5}$$

Finally, we add the *demand per location* to the state. This is not the demand for the current period, because the current demand is assumed unknown. The demand per location represents the demand in a number of past periods. How many past periods to include here is interpreted as a hyperparameter that can be tweaked. The number of demand periods taken into account should allow deducing the recent trend of the demand, but an excess amount of demand periods might confuse the algorithm. Per demand period taken into account, we record the demand per product and location. The values for the shops represent the external demand, per time step, and product. The demand for the factories represents the number of products shipped from this factory, summed over all shops, in the time step in question. Thus the demand per location becomes a tensor consisting of X stacked matrices of the number of products by the number of locations. For our example with one factory, three shops, two products and three past demand periods, this looks like:

$$D = \left[\begin{bmatrix} D^{t-1}_{0,0} & D^{t-1}_{1,0} & D^{t-1}_{2,0} & D^{t-1}_{3,0} \\ D^{t-1}_{0,1} & D^{t-1}_{1,1} & D^{t-1}_{2,1} & D^{t-1}_{3,1} \end{bmatrix} \begin{bmatrix} D^{t-2}_{0,0} & D^{t-2}_{1,0} & D^{t-2}_{2,0} & D^{t-2}_{3,0} \\ D^{t-2}_{0,1} & D^{t-2}_{1,1} & D^{t-2}_{2,1} & D^{t-2}_{3,1} \end{bmatrix} \begin{bmatrix} D^{t-3}_{0,0} & D^{t-3}_{1,0} & D^{t-3}_{2,0} & D^{t-3}_{3,0} \\ D^{t-3}_{0,1} & D^{t-3}_{1,1} & D^{t-3}_{2,1} & D^{t-3}_{3,1} \end{bmatrix} \right]. \tag{3.6}$$

All components of the state can be stacked into one large tensor to represent the environments state in a certain time step.

3.2.2 The action space

The action space contains the range of possibilities in which the agent can influence the state of the environment. These are the inputs to the environment that belong to the control sphere of the agent. The goal is for the agent to select the possible action leading to the highest expected reward, in each time step. There are two different ways the agent can influence the state of the environment: by choosing to produce products and by choosing to transport products from a factory to a shop. We call these the production action and the transportation action respectively. Every time step a production and transportation action are chosen. It is also possible to choose zero-actions, which corresponds to not acting.
The production action consists of defining one production amount per product, per factory in every time step. In our example with one factory and two products this looks as follows:

$$P = \begin{bmatrix} P_{0,0} \\ P_{0,1} \end{bmatrix}. \tag{3.7}$$

The transportation action consists of deciding how many products to ship from every factory to every shop, for each product. In the case with one factory, three shops and two products this would look as follows:

$$S = \left[\begin{bmatrix} S_{0,1,0} \\ S_{0,1,1} \end{bmatrix} \begin{bmatrix} S_{0,2,0} \\ S_{0,2,1} \end{bmatrix} \begin{bmatrix} S_{0,3,0} \\ S_{0,3,1} \end{bmatrix} \right]. \tag{3.8}$$

Where $S_{f,s,p}$ represents the number of products of type p to be transported from factory f, to shop s.

3.2.3 The reward function

The definition of the reward function is one of the most important elements of any reinforcement learning application. This is no different in the case of inventory optimisation. The reward function is the signal that steers the behaviour of the agent. Only what is measured in the reward will be reflected in the agent's behaviour. This provides certain freedom to the RL approach for inventory management: it allows companies to specify the reward functions according to the criteria they value the most. Values differ strongly among companies, discount retailers and high-end speciality shops probably look at certain criteria in different ways.
As mentioned in the introduction we judge the agents' performance based on the achieved profit for the complete supply chain. Which means an attempt to maximise the attained profit. However, components like the service level achieved could also be included in the reward function, to aim for a different outcome. In the remainder of this subsection, the components of the demand function will be mathematically specified.

Revenue

The revenue generated in a time step is defined as the number of products that are sold in that time step times their price, summed over all shops and products. The number of products sold in a shop, in a time step, is equal to the number of products demanded and available in the shop:

$$\varrho = \sum_{s=1}^{n} \sum_{p=0}^{m} \max(I_{s,p}, D_{s,p}) \times \rho_p, \tag{3.9}$$

where ϱ is the revenue, $I_{s,p}$ and $D_{s,p}$ are the inventory level and external demand respectively, in shop s for of product p, and ρ_p is the price per product p.

Production cost

The production cost in a time step is equal to the number of products that start production in that time step times their unit production cost, summed over all factories and products:

$$\Pi = \sum_{f=0}^{o} \sum_{p=0}^{m} P_{f,p} \times \pi_{f,p}, \tag{3.10}$$

where Π is the production cost, $P_{f,p}$ is the production action per factory per product, and $\pi_{f,p}$ is the unit production cost per factory per product.

Holding cost

The holding cost in a time step is defined as the number of products in inventory times the cost of holding a unit in inventory, summed over all locations and products:

$$H = \sum_{l=0}^{n} \sum_{p=0}^{m} I_{l,p} \times \eta_{l,p}, \tag{3.11}$$

where H is the holding cost, and $\eta_{l,p}$ is the unit holding cost per location per product.

Transportation cost

The cost of sending products from one location to another is calculated per truck. There is a fixed cost per truck needed, irrespective of the number of products in the truck. The transportation cost in a time step then becomes the number of required trucks times the cost per truck, for all product location combinations:

$$T = \sum_{f=0}^{o} \sum_{s=1}^{n} \left\lceil \frac{\sum_{p=0}^{m} S_{f,s,p}}{C_p} \right\rceil \times \tau_{f,s}, \tag{3.12}$$

where T is the transportation cost, $S_{f,s,p}$ is the transportation action per factory, shop and product. Where C_p is the truck capacity per product, and $\tau_{f,s}$ is cost of sending a truck from factory f to shop s.

Setup cost

The setup cost is a fixed cost per factory and per product, for both production and transportation. These are incurred when a product is produced in a factory in a given time step and when a product is transported from a factory in a given time step. This results in:

$$K = \sum_{f=0}^{o} \sum_{p=0}^{m} (\iota_{f,p}^{\mathrm{P}} \times \kappa_{f,p}^{\mathrm{P}} + \iota_{f,p}^{\mathrm{T}} \times \kappa_{f,p}^{\mathrm{T}}), \tag{3.13}$$

where K is the setup cost, $\iota_{f,p}^{\mathrm{P}}$ and $\iota_{f,p}^{\mathrm{T}}$ are the indicator variables indicating when a production and transportation setup cost are incurred. The setup costs for for production and transportation are given by $\iota_{f,p}^{\mathrm{P}}$ and $\iota_{f,p}^{\mathrm{T}}$.

Opportunity cost of shortage

The opportunity cost of shortage is equal to a penalty for every unit demand that could not be met with the inventory, summed over all shops and products. These are the products that might

be back-ordered in other cases. The opportunity cost of shortage represents a loss of goodwill from the customer, i.e. they might be more inclined to go to a competitor in the future and the opportunity cost of sales. From which we derive:

$$\Omega = \sum_{s=1}^{n} \sum_{p=0}^{m} min(I_{l,p} - D_{l,p}, 0) \times (-1) \times \sigma_{l,p}, \tag{3.14}$$

where Ω is the opportunity cost of shortage and $\sigma_{l,p}$ is the penalty considered per product.

3.3 Improving Inventory Optimisation with Reinforcement Learning

As discussed in the section on existing solutions to the problem of inventory management. We intend to improve the available tools for inventory optimisation along both its paradigms.
In statistical terms, the EOQ-based models can be seen as parametric models, where we explicitly define assumptions for the underlying phenomena, in this case the demand. The RL approach can be seen as a non-parametric counterpart, making no assumptions about the underlying principles, but needing a lot more data to achieve good results. As a consequence EOQ-based models can used only in a limited set of case, while the RL based method can be used for different problem specifications.

3.3.1 The choice of the lot size

The current choice of the lot size is based on the EOQ theory. While basic EOQ provides a sound place to start, today's more complex reality often requires more complex mathematical variants of EOQ to be able to calculate the optimal lot size. As we have seen in the section on variants of the EOQ models, each slightly different situation requires a different kind of EOQ model. Because of this find the correct EOQ model is not possible for most realistic situations. We propose to use a data-driven approach such as Reinforcement Learning to find a near-optimal approach without having to make any of the assumptions necessary with EOQ.

3.3.2 Determining the reorder point

Improve the choice of the reorder point in two ways. First of all, rather than assuming the distribution of the reward, we can take a more data-driven approach. By learning from experience from the actual past demand data, it is possible to train an agent to determine the reorder point. Without having to pass it the assumed demand distribution. Secondly, it is no longer necessary

to assume the distribution of the demand always remains constant. Both the demand rate and its variance can change over time. Using the data-driven approach an agent can learn that the demand and its uncertainty are different during different times of the year. By letting the agent choose when to place an order and when not to, the reorder point is being set dynamically.

Chapter 4

Experiments with Discrete Action Spaces

To further explore the possibilities of reinforcement learning in the domain of inventory optimisation, a problem similar in size to the one discussed in Kemmer et al. [18] is regarded. Kemmer et al. [18] proved that a policy gradient RL method using a quadratic or RBF based function approximator is able to outperform a (Q,R)-heuristic policy in terms of maximising the supply chain profit, in a case with perturbed sinusoidal demand in three shops, supplied by one factory. We note that this is a non-trivial achievement, as only two out of four of the tested methods succeeded in this endeavour.

In the following two chapters we build up complexity through experiments by first starting with a discrete action space as in Kemmer et al. [18]. We refer to this problem as the discrete base case. Next, we attempt to use the same method on a problem with a larger action space, this problem is referred to as the expanded case. Finally, we attempt to solve the problem from the first setup, using a continuous action space instead of a discretised one. This is referred to as the continuous base case.

The supply chain at the foundation of the base case consists of one factory, three shops and one product. We consider four types of stochastic demand scenarios for the experiments regarding the base case, with varying levels of complexity. Comparison of the performance of RL agents with the benchmark across these demand scenarios should allow us to form an answer to research questions 1.1 and 1.2. The expanded case should indicate the abilities of RL agents in terms of inventory management, concerning problems characterised by a large discrete action space.

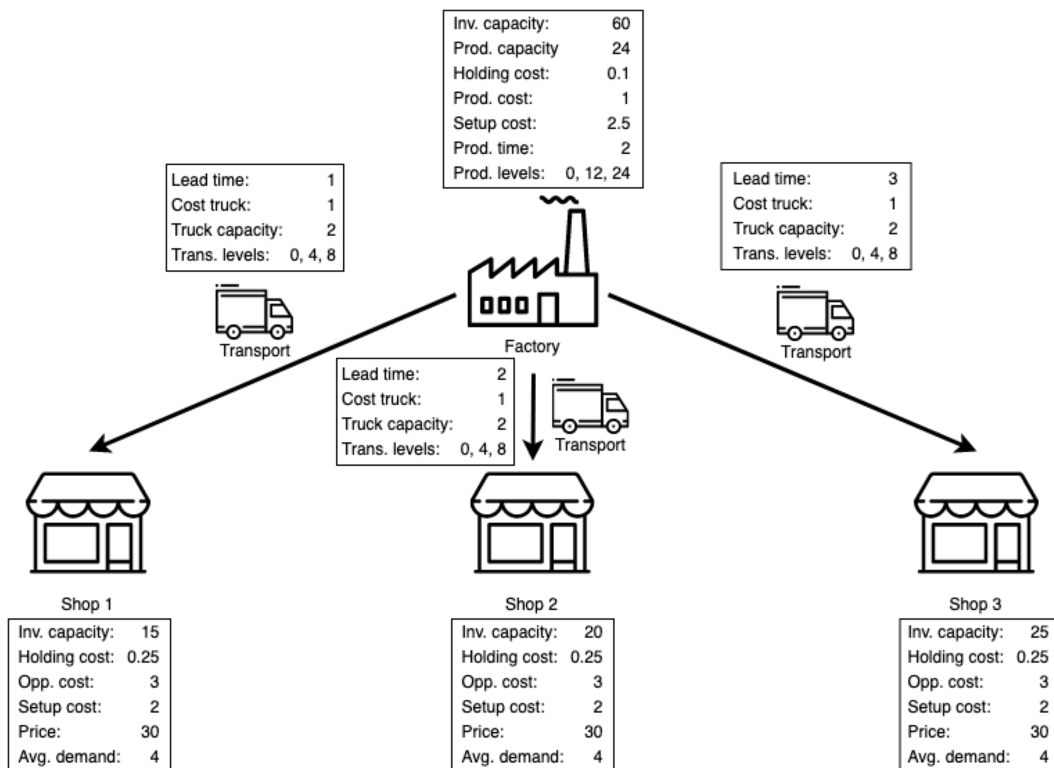


Figure 4.1: An overview of the environment used for the base case using a discrete action space. The environment parameters are included per component of the environment. Inv., prod., and trans. refer to inventory, product or production, and transportation respectively.

The agents used in the cases with a discrete action space are both a combination of NFQ and DQN. We use NFQ enhanced with the extensions made for DQN such as prioritised experience replay, as mentioned in section 3.1.2. The agents used in the cases with a continuous action space are DDPG agents, as presented in section 3.1.3.

4.1 Base case: Discrete approach

4.1.1 Experiment

As mentioned in the introduction of this chapter, the base case we discuss consists of a supply chain with 1 factory, 3 shops and 1 product. In this work, we categorise the environments by these metrics. We will refer to a case with α factories, β shops and γ products as an (α,β,γ)-case. Thus the base case is a (1,3,1)-case.

Action and state space

When defining the action space for this experiment as done in section 3.2, we see that the action space has four dimensions. One for the production of products and three for the transportation of the product from the factory to each of the three shops. In the experiments using a discrete action space, we have decided to give the agent three values to choose from along each dimension. Here this corresponds with $3^4 = 81$ action possibilities in every time step. As discretising the action space considerably limits the action space, it is important to make sensible decisions with respect to which values to include, which is why we base these values upon the EOQ theory. For the discrete RL approach, the agent can choose between not performing a certain action, performing an action, producing or transporting half of the EOQ amount or performing an action, producing or transporting the EOQ amount. For the sake of symmetry, it seems logical to include a fourth value with 1.5 times the EOQ amount, however, this would bring the size of the action space up to 256 different actions. We believe the advantage of this additional freedom for the agent would not weigh up against the additional complexity. We choose to include 0.5 rather than 1.5 times or more times the EOQ amount in the spirit of JIT-management.

Comparing this action space, 81 distinct actions, to other famous discrete problems tackled for example by DQN we see that this is already a rather large action space. Two of the most well-known examples are probably 'cart pole' and 'mountain car' from the OpenAi gym [6]. These have discrete action spaces with only two and three different actions respectively. An example of a more complex environment with a discrete action space can be found in the Arcade Learning Environment (ALE) [3]. This collection of Atari games has an action space of 18 distinct actions.

There are two problems with using large discrete action spaces. On the one hand, it becomes exceedingly difficult to have enough examples of all different actions. On the other hand, there are no relations between actions. For example, the action of not producing would be as similar to producing the minimum amount as to producing the maximum amount. This makes it more difficult to learn a certain behaviour.

When defining the state space as done in section 3.2 for this (1,3,1)-case, we see that it has 20 dimensions. Four dimensions for the inventory at each of the locations, four for the work in process and the products in transit to each of the shops and since we use the demand of the past 3 time steps in this experiment, another 12 for the past demand. Compared to the smaller problems mentioned above, 'cart pole' and 'mountain car' this state space is much more

high-dimensional. These have four and two dimensions respectively. Note that they might have more fine-grained steps along each of these dimensions. On the other hand, it must be noted that these two toy environments are fully observed and do not have inherent randomness to them, which makes these problems a lot simpler. The Arcade Learning Environment [3] is more similar to our environment here in that perspective. Compared to the state space used for ALE, our state space is rather small. The state space used by Hausknecht et al. [12] contained up to 80 features for the location of all objects. While in Mnih et al. [22] raw pixel data was used 84 x 84 sized grayscale images, leading to much higher dimensionality.

Demand function

As mentioned in the introduction of this chapter we use four variants of the same demand function to represent varying levels of closeness to representing reality. The basic demand function is a stationary sinusoidal function with constant variance. This is the type of demand function best suited for the heuristic policies. In this case the variance V and trend M in equation 4.1 are set to zero.

$$D_{t,s} = \left\lfloor \frac{D_s^{\text{avg}}}{4} \times V^{\frac{t}{t_{\text{total}}}} \times \sin\left(\frac{2(t+2l)}{26}\right) + D_s^{\text{avg}} + M \times \frac{t}{t_{\text{total}}} + \epsilon_{t,s} \right\rfloor \tag{4.1}$$

We relax assumptions, creating three more demand functions: a demand function stationary in terms of the mean but with non-stationary variance, a demand function non-stationary in terms of the mean with stationary variance, and a demand function non-stationary in terms of both the mean and the variance. As the heuristic method is not able to perceive this change and adapt itself, we expect the RL method to gain an advantage with the increasing complexity of the demand function. The situation is similar to reality as when a heuristic inventory policy such as (Q,R) or (s,S) is used, the calculations will not be redone continuously. It seems more realistic that these would be redone periodically or when performance is perceived to deteriorate.

In the stationary and homoskedastic case the demand fluctuates between 4 and 6 units of demand per shop. It can be seen in equation 4.1 that the seasonality of the demand is slightly shifted between shops. Random perturbations are also different for every shop. When the trend factor is included in the demand function, every new episode a trend is randomly chosen within $[\frac{-D^{\text{avg}}}{2}, \frac{D^{\text{avg}}}{2}]$. This leads the demand to transition to a value between 50% and 150% its starting

value over the course of one episode. When the variability change factor is included, every new episode a variability change is chosen within $[\frac{1}{3}, 3]$. This amplitude of the sine function is multiplied with this factor to change the variability. The variance factor gets a power that increases from zero to one, to let the variance increase stepwise. Note that the variability change factor is sampled in such a manner that we should on average have as many episodes with decreasing as increasing variability over time.

(Hyper)parameter setting and setup

The parameter settings of the environment can have a big influence on the outcome of the environment, but within a logical range, we do not believe them to have much influence on the analysis of the outcome. We confirm the supply chain can attain a profit to avoid any perverse incentives. An overview of the experiment and parameter settings can be found in figure 4.1.

Here we still wish to discuss the *hindsight* and the *length of an episode*. With hindsight we refer to the number of periods of past demand included in the state. In this experiment, the hindsight is chosen as 3. Several setting were tested for this parameter, but including more steps did not seem to have a positive effect on the performance of the agent. It seems critical to have a minimum of three time steps to estimate the first and second-order derivative of the demand function, which corresponds to the change in demand and the rate at which it is changing. On the other hand, including more periods of past demand might confuse the agent if it does not add more information. The length of an episode is set to 52 in the experiments we conduct. While in this case, one time step seems to represent one week in a year, we want to stress that this is not necessarily important. Each time step in this simulation could just as well represent one hour or even minute in reality (given some slight changes in the parameter setting). We chose for an episode length of 52 steps because a shorter episode length allows for shorter training times. Otherwise, we might, for example, have chosen for 365 steps in an episode, without considerable change in the outcome of the analysis. It was, however, important to set a certain episode length, for it helps the agent to converge on a solution. The reward in the final step of an episode forms a fixed anchor point for the calculation of the Q-values, using the Bellman equation.

We chose to use the DQN implementation by Stable Baselines for our RL agent. The choices made for the hyperparameters were based on Mnih et al. [22] and Riedmiller [25] and the default values suggested by Hill et al. [16]. From there improvements were sought by changing

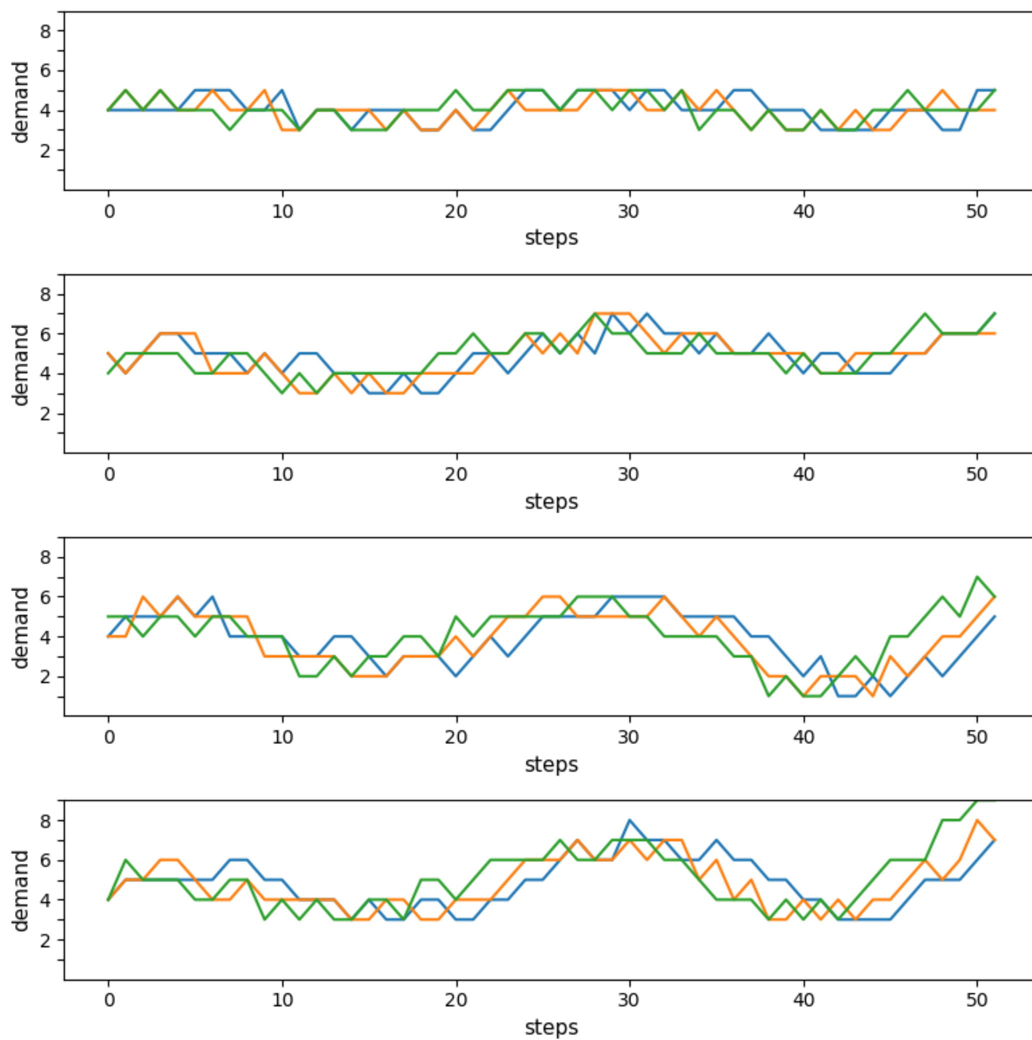


Figure 4.2: Samples of the demand functions used for the base case. From top to bottom these are: stationary in mean and variance, non-stationary mean and stationary variance, stationary mean and non-stationary variance, and non stationary mean and variance.

parameters in both directions, one at a time. Keeping the setting with the best results. For the function approximator we base ourselves upon Riedmiller [25]. Riedmiller [25] uses a network with two hidden layers of five nodes for an action space with two choices. We extend this reasoning two our action space and choose the power of two closest to this value, resulting in an MLP with two hidden layers consisting of 256 nodes. The agents were trained for a total of one million time steps each, which corresponds to 10,000 to 15,000 completed episodes. We found the best results with a rather small learning rate of 0.0001. As expected we found that a relatively small discount factor γ gave the best results: 0.92. This seems logical because the environment provides very frequent rewards, where some other environments only provide one reward per episode. During the first 900000 steps, the exploration factor ϵ is linearly reduced from 100% to 1%. We use prioritised experience replay with the default parameters suggested by Schaul et al. [27]: a prioritisation factor α of 0.6 and an annealing factor β of 0.4. Finally, mini-batches of 32 steps are used for training.

4.1.2 Results and Discussion

In Table 4.1 we display the summarised results for the discrete approach to the base case. As the random component in the demand can lead to non-negligible differences across episodes, the measures were divided by the demand in its episode to make the results more comparable, both across agents and over time. The displayed figures for both the RL approach as the heuristic method are averages over 1000 episodes. For the heuristic approach, the trade-off method was used to set the reorder point, using the same costs as in the RL case. A more comprehensive version of table 4.1, including the standard deviations for all measures can also be found in the appendix.

When studying the results in Table 4.1 there are three most notable results. The first is that the RL agent is able to outperform the heuristic agent for all four of the conducted experiments. We note that the performance of the RL agent is always more consistent than that of the baseline because the standard deviation over 1000 experiments is smaller. The difference between the average reward attained by the RL agent and the (Q,R)-agent is more than twice the standard deviation of the RL agent's reward and more than once the standard deviation of the (Q,R)-agent's reward.

Demand	Agent	Reward	Revenue	Prod.	Hold	Trans.	Setup	Opp.
Stationary mean	NFQ	26.76	28.95	0.99	0.32	0.48	0.29	0.11
and variance	(Q,R)	25.23	27.38	0.92	0.31	0.46	0.19	0.26
Non-stationary	NFQ	26.82	29.13	1.01	0.40	0.49	0.32	0.09
variance	(Q,R)	24.57	26.77	0.90	0.34	0.45	0.19	0.32
Non-stationary	NFQ	25.96	28.30	0.98	0.40	0.48	0.32	0.17
mean	(Q,R)	24.45	26.72	0.91	0.37	0.45	0.20	0.33
Non-stationary	NFQ	26.23	28.63	1.00	0.47	0.48	0.32	0.14
mean and variance	(Q,R)	24.15	26.41	0.90	0.36	0.44	0.19	0.36

Table 4.1: Results for the base case, using a discrete action space. The NFQ agent outperforms the (Q,R) benchmark method across all demand scenarios. We note that the advantage is due to higher product availability instead of decreased costs.

Second, as expected we see that the profit generated by both agents decreases with the increasing complexity of the reward function. As expected the RL agent has a larger advantage over the heuristic method in the more complex demand scenarios, compared to the base demand scenario. The RL agent loses less of the attained reward in the more complex scenarios compared to the heuristic agent.

Lastly, we note that the comparative advantage of the RL agent over the heuristic benchmark based on the trade-off method from section 2.1.3, is due to higher product availability rather than saving expenses on the production, transportation or holding of products. An agent able to meet all external demand would achieve a revenue equal to the price of the product, which is 30 in this case. This enables us to calculate the percentage of the total demand the agent was able to satisfy. Across the different demand scenarios, the RL agent is able to serve 96.67% to 94.27% of all external demand, averaged over 1000 episodes. The (Q,R)-agent is able to serve between 91.30% and 88.27% of all demand. At the same time, the RL agent incurs more costs than the (Q,R)-agent. The largest relative differences are found for the setup cost, where the RL agents' costs are over 50% higher than the (Q,R)-agents' costs. The RL agent also holds more inventory and produces more products than the heuristic agent, on average. The latter might be an indication that the profit attained by the (Q,R) agent could be increased by increasing

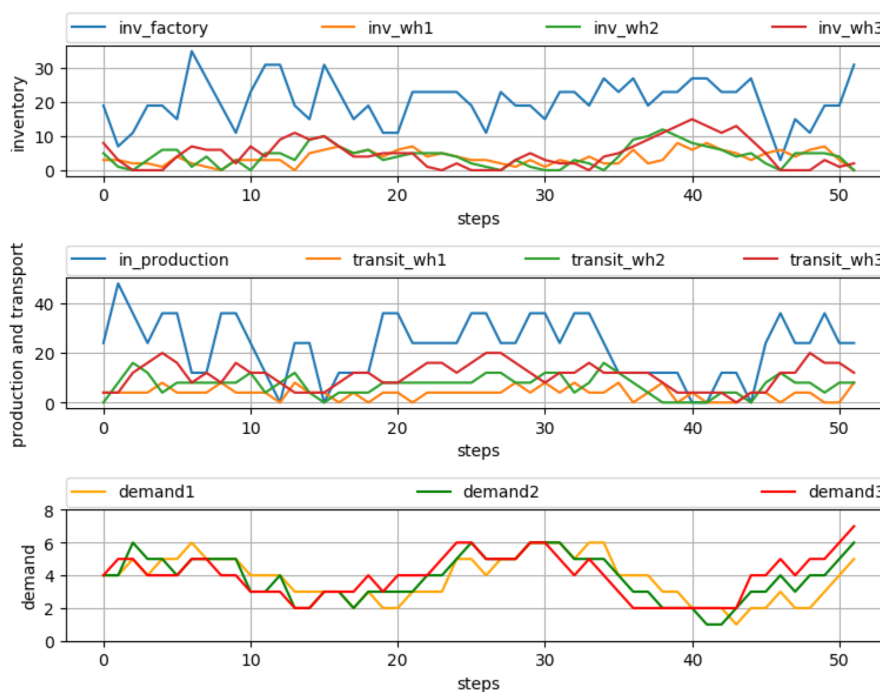
the held safety stock. This could be done by increasing the reorder point. The outcome would be a trade-off between the extra costs of holding products and the increased revenue.

We tested the hypothesis that an increase of the reorder point could lead to increased reward for a (Q,R)-agent. To test this, we set the reorder point a little higher to the maximum inventory minus the EOQ amount. Using a higher reorder point is not possible for this environment, because the maximum inventory refers to the inventory position, indicating both finished goods and work in process or transportation. The test was conducted for both the case where the demand is stationary in mean and variance and the case where they are non-stationary.

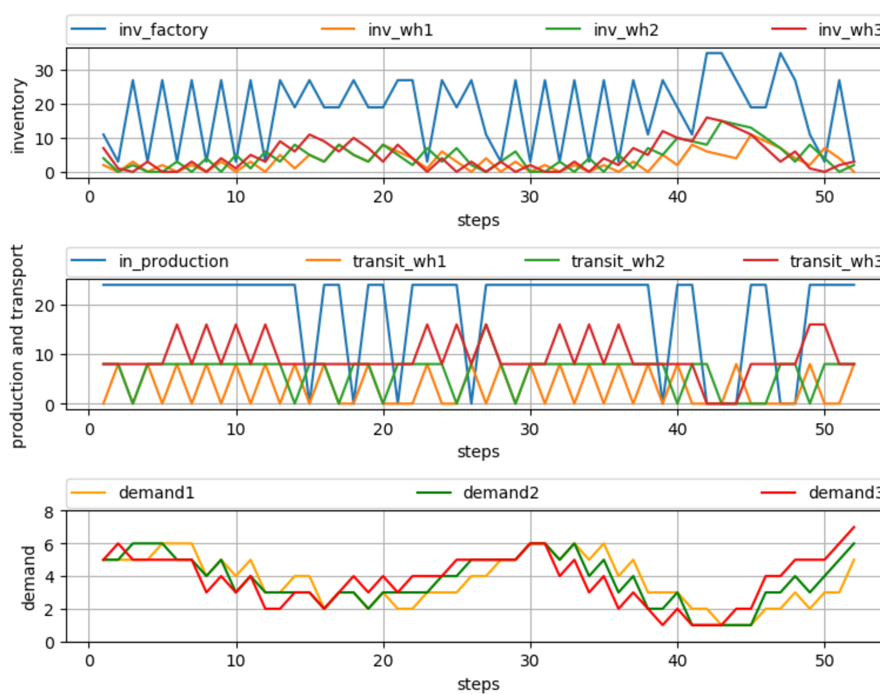
For both cases, we find a slight improvement of the reward of the (Q,R) agent. The performance in the simplest case improved by 2.77% which is the result of a 2.85% increase in the revenue and a 26% increase in the holding costs. The non-stationary case knew a reward improved by 2.11% arising from a 2.35% improvement in the revenue and a 12% increase in holding costs. Although this is an improvement, these results still leave a confident margin in the advantage of the RL agent.

Figure 4.3 depicts the evolution of the inventory level, the work in process and transportation, and the demand during one episode for both the RL agent and the benchmark method. These figures give an insight into the behaviour the RL agent has learned, compared to the well-known behaviour of the (Q,R) inventory policy. Conform what can be found in table 4.1, the inventory level is in the same range for both agents. The inventory level of the RL agent knows a little less extreme fluctuations than that of the (Q,R) agent. Both agents show similar behaviour in terms of inventory build-up in the shops. Inventory levels increase during periods of low demand and disappear quickly in periods of high demand. It is more likely that the inventory builds up because the demand is below average, rather than that this is a deliberate process to build up a buffer for the coming period of high demand. The RL agent responds to the varying demand mainly by decreasing the products in production and the products transported in periods of low demand, both in terms of frequency and amount. While the (Q,R)-agent produces and transports less often, but still produces the EOQ amount. It is clear that the RL agent has learned to execute different actions in different situations, instead of repeating the same action over and over again.

Keeping track of which exact actions are taken in every step provides a more detailed understanding of the agents' behaviour. For the NFQ agent in the base case in the stationary and



(a) RL agent



(b) (Q,R)-agent

Figure 4.3: Inventory evolution during one episode for the base case, using discrete action space. Both methods have similar behaviour in terms of inventory build up. Overall the RL agent holds more inventory than the (Q,R)-agent and the inventory level in the shops is reduced to zero for shorter periods. The RL agent responds to the demand function in a reactive manner by producing more in periods of high demand and decreasing production in periods of low demand.

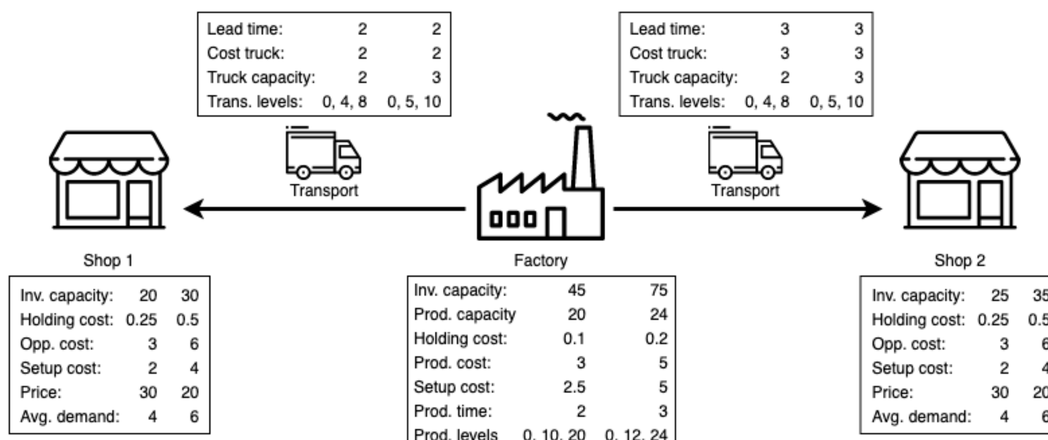


Figure 4.4: Expanded case: One factory, two shops and two products, using a discrete action spaces. The parameters listed in the first column of each table refers to product 1, those in the second columns refer to product 2.

homoskedastic demand scenario, we find that 58 of the 81 available actions are used over a course of 1000 episodes. The 39 actions used most frequently are used in more than 99.5% of all steps. For the (Q,R)-agent we find that less of the available action space is used for the same case. Of the available 81 actions, only 19 are used by the heuristic. With 15 of these actions being used in more than 99.8% of all steps. For the (Q,R)-agent we see that the use of transporting half of the EOQ amount is indeed minimal. This seems to confirm that the agent does not experience undue inventory shortages in the factory. The (Q,R)-agent will never produce half of the EOQ amount because resources for production are not restricted in this case.

4.2 Expanded case: Discrete approach

4.2.1 Experiment

In this section, we discuss a slightly larger problem. The supply chain we use here has 1 factory, 2 shops and 2 products. Further in the text, we will refer to it as the expanded case or the (1,2,2)-case.

Action and state space

When defining the action space as done in section 3.2, it can be found that this action space has 6 dimensions. Each product has one dimension for production and two dimensions for transportation. While in these terms, this is only 50% larger than in the previous section,

this brings the action space up to $3^6 = 729$ distinct actions. Which makes this action space 9 times larger than the one for the discrete base case. The three levels are no action, producing or transporting half of the EOQ amount, or producing or transporting the EOQ amount. Note that introducing the fourth action with 1.5 of 2 times the EOQ amount for every dimension would lead to an action space with 4096 actions in this case. Still a discrete action space with 729 actions is hardly comparable to the ones for 'cart pole', 'mountain car' and even ALE discussed in the previous section. We expect the agent to have a lot of difficulty with the exploration of the action space.

For the expanded case, the state space has a total of 30 dimensions. Six for both the inventory and the work in process and transit, and 18 for the past demand. For non-image data, this is a rather large state space. For comparison, some of the MuJoCo environments [32] (e.g. humanoid) also have a state space with around 30 DOF. The toy environments like 'pendulum' and 'acrobat' have 2 and 6 dimensions respectively [6]. It should, however, be noted that our inventory management environment uses integers only, which makes it less fine-grained than other continuous state spaces.

Demand function

We test the expanded case on the same type of demand function as the base case. The focus in on the scenario with stationary and homoskedastic demand. We verify whether similar performance can be attained on a problem with a larger state and action space, using the discretized action space approach.

(Hyper)parameter setting and setup

The setup and hyperparameter setting for the extended problem is very similar to that of the base case. Both hyperparameters for the environment and the agent are based upon those found to work best in the base case. The *hindsight* is set to three. Following the reasoning of the discrete base case, discussed in section 4.1.1: the *number of steps per episode* is set to 52. An overview of the experiment can be found in figure 4.4.

There are three main differences in the hyperparameter setting between the discrete base and expanded case. The first is the size of the network. We extend the reasoning used in the base case: the power of two closest to 2.5 times the number of distinct actions. This results in an MLP with two hidden layers with each 2048 nodes. Because the expanded case is more difficult

Demand	Agent	Reward	Revenue	Prod.	Hold	Trans.	Setup	Opp.
Stationary mean	NFQ	15.43	21.98	4.03	0.54	1.01	0.60	0.36
and variance	(Q,R)	11.47	17.41	3.22	0.46	0.80	0.30	1.14

Table 4.2: Results for the expanded case, using a discrete action space. Compared to the base case with discrete actions the difference between the NFQ-agent and (Q,R)-agent has increased across all reward components. The RL agent strongly outperforms the (Q,R)-agent.

than the base case and training a larger network takes more time, the agent was trained for four million time steps. The exploration rate ϵ is decreased over the first 90% of the training steps, which means the first 3.6 million steps in this case. For the expanded case we find better results using a smaller learning rate: the learning rate is halved compared to the discrete base case, being set to 0.00005. The other hyperparameters remain fixed across the discrete base and expanded case.

4.2.2 Results and Discussion

The results for the discrete approach for the expanded case can be found in table 4.2. As for the base case, the reward and reward components are expressed as values per unit demand, averaged over 1000 episodes. For the expanded case, however, there are two different products.

The analysis of a system characterised by multiple products becomes more complex. To achieve a detailed understanding of the behaviour learned by the agent it is necessary to keep track of reward and its components for each product separately. In this study we do not track these product level reward metrics, thus we focus on a macro-level analysis of the behaviour of both agents. From the environment parameters, we know that on average 40% of the total demand are products of type 1. This allows a high-level analysis of the results.

The first observation is that the RL agent outperforms the (Q,R)-agent by a large margin. We noted that the relative advantage of the RL agent over the benchmark is larger for the expanded case than in the smaller discrete base case. Contrary to our expectation the RL agent's performance does not seem stifled by the larger discrete action space. In general, we find that the advantage of the RL agent comes from the higher attained product availability. The RL agent sustains higher production, holding, transportation and setup costs than the (Q,R)-agent.

An agent able to meet all external demand would attain a revenue close to $0.4 \times 30 + 0.6 \times 20 = 24$, with some margin for the random character of the demand. This indicates that the agent is able to satisfy 92% of the demand. The (Q,R)-agent satisfies 71% of all demand. Both of the values are under the assumption that both products are sold proportionally to the demand by both agents. We find that these values are slightly lower than those found for the base case. While this was expected for the RL agent because of the increased problem size, the strongly decreased performance of the (Q,R)-agent is surprising because of the decentralised manner in which the solution is found.

The higher costs incurred by the RL agent compared to the benchmark method are similar to the base case. Yet we observe that the difference between the agents, in this case, is larger than in the discrete base case. This might be an indication that the RL agent uses a different strategy than in the smaller case.

To achieve a better understanding of the agents' behaviour in the expanded case, the actions used by the agent can be studied as done for the base case. The RL agent uses 49 of the 729 available actions. This means that less than 7% of the action space is used. This figure was 73% for the RL agent in the base case. The (Q,R)-agent uses 60 of the available actions, corresponding to about 8% of the action space. While the difference is less striking compared to the difference found for the RL-agent, it is still striking. This raises the question of whether improved performance could be achieved by limiting the action space to two values per dimension: not acting, or producing or transporting the EOQ amount.

Chapter 5

Experiments with Continuous Action Spaces

This chapter removes one constraint from the experimental setup in chapter 4, i.e. the action space is now a continuous action range for each action dimension instead of a combination of one of three discrete values for every action dimension. While increasing the complexity of the problem, this is expected to enable solving larger problems. Because NFQ and DQN are not suited for problems with continuous action spaces, another algorithm is needed. We opt to use DDPG. This algorithm was posed by Lillicrap et al. [20] to be the equivalent of DQN for continuous action spaces. The similarity between DQN and NFQ on the one side and DDPG on the other side is that they all use value learning based on Q-learning. Which is supposed to make them more sample efficient [20, 22].

Even though using a continuous action space allows us to work with larger action spaces because of the different representation, this also makes the problem a lot more difficult. This is why we start by tackling an equivalent of the discrete base case using a continuous action space, rather than starting with a larger problem.

5.1 Experiment

An overview of the continuous approach of the (1,3,1)-case can be found in figure 5.1. As in the discrete case, we will regard four different demand scenarios with varying levels of complexity.

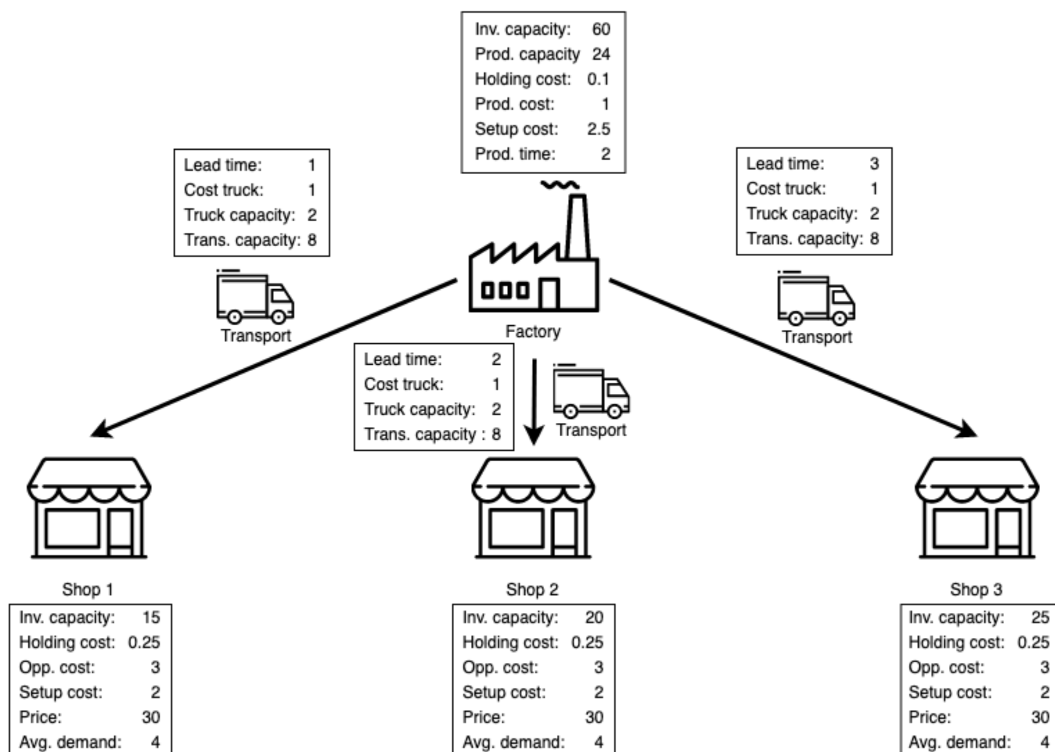


Figure 5.1: An overview of the environment used for the base case using a continuous action space. The environment parameters are included per component of the environment. Inv., prod., and trans. refer to inventory, product or production, and transportation respectively.

5.1.1 Action and state space

The action space for this experiment has four dimensions. For each of the dimensions, an action is selected ranging from zero, or not acting, to producing or transporting the EOQ amount (as defined in section 4.1.1). Limiting the action space to the EOQ amount has the benefit to improve comparability between the results found in chapter 4 and here, for future research however it might be more interesting to give the agent access to a larger action space. The state space for this experiment is the same as discussed for the discrete base case.

5.1.2 Demand function

The demand functions used for the continuous base case are the same as those discussed in section 4.1.1. The different scenarios are (in order of increasing complexity): stationary and homoskedastic demand, non-stationary and homoskedastic demand, stationary and heteroskedastic demand, and non-stationary and heteroskedastic demand.

5.1.3 (Hyper)parameter setting and setup

The setup for this experiment was very similar to the discrete cases, we used the DDPG implementation by Stable Baselines [16], as discussed in section 3.1.3. Compared to other RL algorithms, DDPG is known to be difficult with respect to hyperparameter tuning. This was confirmed by our experience. For the discrete case, it was possible to attain good results based on values suggested in the introducing papers. Attempts to do the same for DDPG lead to poor performance.

To tackle the problem of hyperparameter tuning a large random hyperparameter search was set up. We recorded the hyperparameters used for DDPG from different sources [14, 15, 17, 19, 20, 24] and selected the minimum and maximum value found for each hyperparameter. The resulting ranges can be found in table 5.1.

The tested hyperparameter settings were found by randomly selecting values from the ranges in table 5.1 using a uniform function. The first two entries in the table are the learning rate for the actor and critic network respectively. Critic L2 regularisation is the weight factor used for L2 regularisation in the critic network. Gamma is the discount rate for future rewards, used in the Bellman equation. Tau is the rate with which the target networks are updated. The values for layers are the number of nodes in the first and second hidden layer respectively.

Hyperparameters	Min	Max
Actor learning rate	0.00001	0.005
Critic learning rate	0.00005	0.005
Critic L2 regularisation	0	0.02
Gamma	0.9	0.999
Tau	0.0005	0.1
Layers	[256,256]	[512,512]
Batch size	32	1536
Buffer size	10000	1000000
O-U sigma	0	1
O-U theta	0	0.5
Reward scale	0.1	1
Layer normalisation	False	True
Binary reward	False	True

Table 5.1: The parameters that were explored during the hyperparameter search and the ranges from which the values were uniformly drawn. The values were found by comparing values found in 5 papers and a blog post [14, 15, 17, 19, 20, 24]

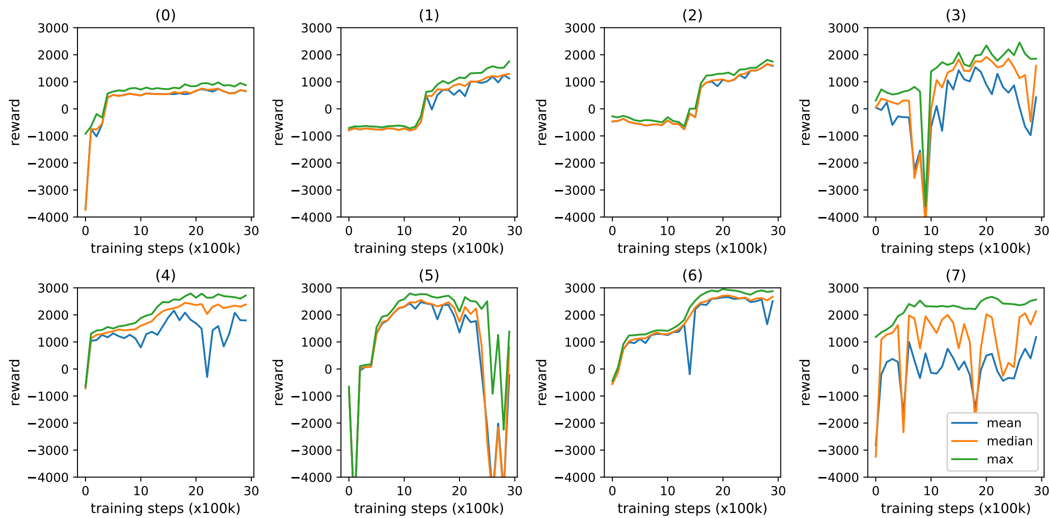


Figure 5.2: DDPG hyperparameter search: training evolution of the best parameter combinations tried in initial run. Plots 4 and 6 give the best impression. Setups 0, 1 and 2 train to slow or a stuck in a local optimum. Setups 3, 5 and 7 are to unstable.

These are set independently of each other. The maximum value for the batch size might seem out of tone, the value is found by taking 1.5×2^{10}. The buffer size is the maximum number of transitions stored for experience replay. The sigma and theta of the Ornstein-Uhlenbeck process are parameters for variance and starting place of this process respectively. The reward scale is a scaling factor with which the reward signal is multiplied. Layer normalisation [2] is a form of normalisation where the input data is normalised across features. The variable binary reward refers to a hyperparameter specific to the environment. If the binary reward is set to *True*, the agent receives a positive reward when a feasible action is selected and a negative reward for unfeasible actions during the first million time steps. This was introduced with the aim of simplifying the task for the agent, because of the highly constrained nature of the action space. This way the agent could start by learning to take feasible actions and later refine this to good actions. This fits within the philosophy of curriculum learning [4].

From the ranges in table 5.1, 160 different hyperparameters settings were randomly drawn. Each separate training run took between 12 and 36 hours to complete. During training, the agents were trained in blocks 100,000 time steps at a time. After 100,000 time steps, the agent was evaluated on an evaluation environment to see performance evolution. Of the 160 tested hyperparameter combinations, 9 were able to achieve a positive positive reward over one episode during evaluation. The best 8 agents attained results between 24% and 86% of the performance

Parameters	Min	Max
Actor learning rate	0.00005	0.0003
Critic learning rate	0.0005	0.004
Critic L2 regularisation	0.002	0.018
Gamma	0.92	0.98
Tau	0.04	0.09
Layers	[256,256]	[512,512]
Batch size	256	650
Buffer size	400000	700000
O-U sigma	0.55	0.7
O-U theta	0.45	0.5
Reward scale	1	1
Layer normalisation	True	True
Binary reward	False	True

Table 5.2: Refined hyperparameter ranges. Based on the hyperparameter settings of the two best performing agents of the initial search, identified in figure 5.2 and 5.3. The new ranges are used to sample new hyperparameter settings for agents. Based on the performance of these new agents, the most interesting regions are narrowed down.

of the baseline during the evaluation, averaged over 100 episodes. The evolution of these 8 agents during training is depicted in figure 5.2.

Figure 5.3 depicts the most important numerical hyperparameters for these 8 agents. For the actor learning rate, it is clear that the best performance is found in the outer left section of the considered spectrum. For the other parameters, the values found for these best-performing agents are still scattered across the whole range. Considering that the agents took into account also still show very diverse performance, this is not surprising. By examining the best performance of each agent averaged over a series of 100 evaluation episodes and the consistency of the evolution of the agents' performance during training, the best-performing agents are selected among these to narrow down the most interesting ranges of the hyperparameter ranges. This lead to the selection of the agents corresponding to plot 4 and 6 in figure 5.2. The hyperparameter values for these two most promising combinations are marked in red in figure 5.3.

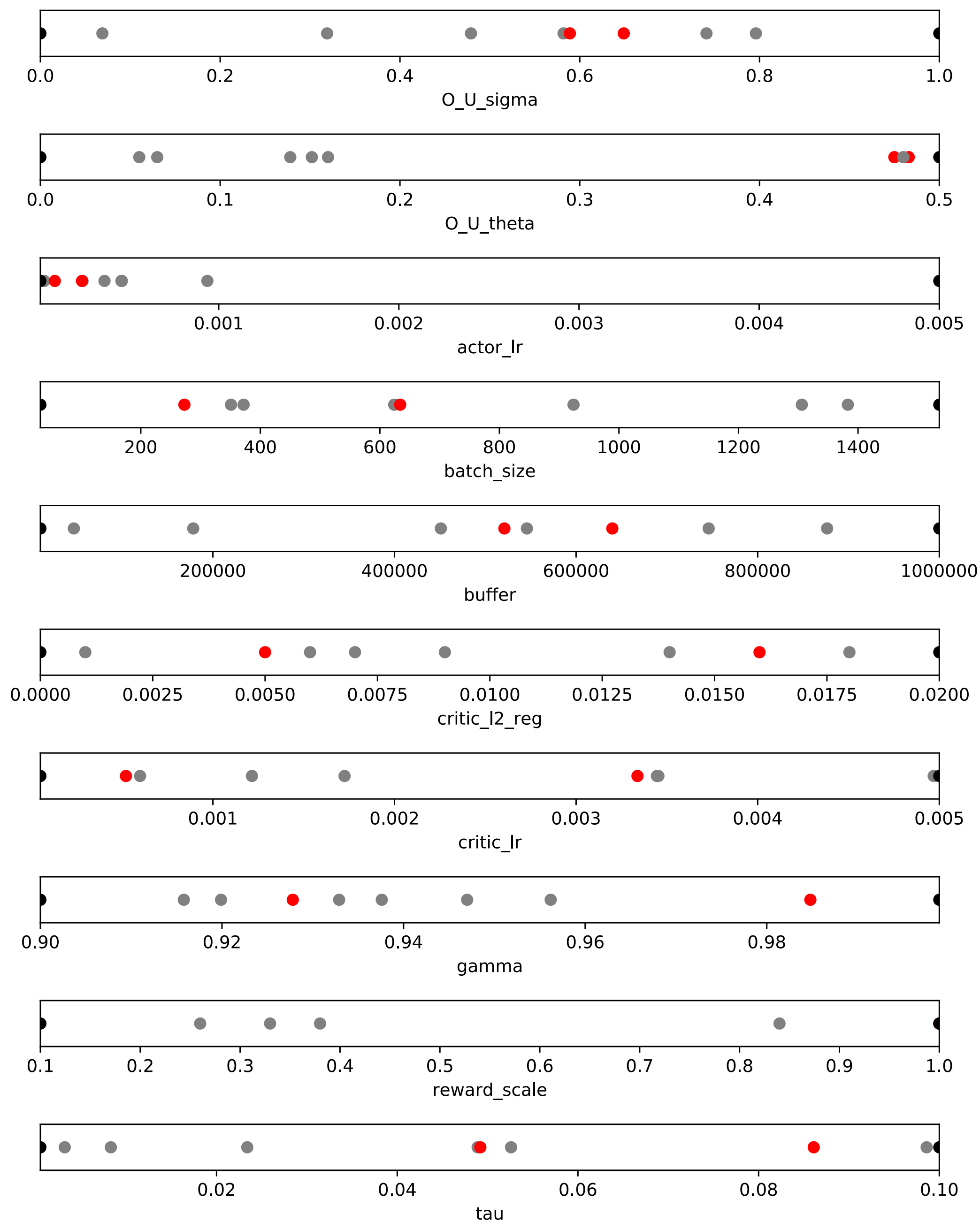


Figure 5.3: DDPG hyperparameter search: hyperparameter settings found to give the best results in the initial round. We see that for most parameters the interesting range is not narrowed down yet. The settings marked in red are the two best performing agents, identified in figure 5.2.

Based on the two most promising hyperparameter combinations marked in red in figure 5.3, a new refined set of hyperparameter ranges was devised. These are listed in table 5.2. Another 40 agents were trained according to the same method as before, using 33 different hyperparameter settings, generated from the revised ranges. 8 agents were trained using the same hyperparameter settings.

Of the 40 agents trained in this second section, only 6 were unable to get a positive reward for one episode during evaluation. This is 15% of the agents, compared to 94% in the initial search. Another 6 agents attained a higher reward than the best agent from the initial 160 agents, averaged over 100 evaluation episodes. While the agents from the second batch performed much better than the initial 160 agents on average, the gain when compared on an individual level was not spectacular. The best agent of the second batch performed 5%-points better than compared to the best of the initial 160 agents, when compared to the baseline, reaching 91% of the baselines reward averaged over 100 episodes during evaluation.

Finally, we note that there was a large discrepancy in the performance of the 8 agents trained with the same hyperparameters. Averaged over 100 evaluation episodes, their rewards are between 43% and 88% of the baseline's reward. This is due to the large importance of random variables in the training of an agent. This has two unfortunate consequences, first of all, it will not be possible to exactly reproduce these agents. Second, we can not rule out the hyperparameter settings that resulted in a bad performance, with full disclosure.

The results of the complete hyperparameter search are summarised in figure 5.4. The circular markers in this figure represent the hyperparameter settings tested. The circle diameter and colour indicate the performance realised by the agent. The categories range from the largest yellow circles for which the best reward was still highly negative, to the smallest dark blue circles for the best-performing agents. The red vertical lines indicate the boundaries from within which parameters were selected during the second batch of the parameter search. The red triangle indicates the hyperparameter setting suggested by Lillicrap et al. [20].

One of the most striking conclusions from this summary is that the hyperparameter values found for this environment often strongly differ from the values suggested by Lillicrap et al. [20]. This confirms the suspicion that DDPG hyperparameters are highly environment-dependent. We find that the agent needed action noise with a higher variance to sufficiently explore the environment, comparable to the value suggested by Plappert et al. [24] for sparse environments.

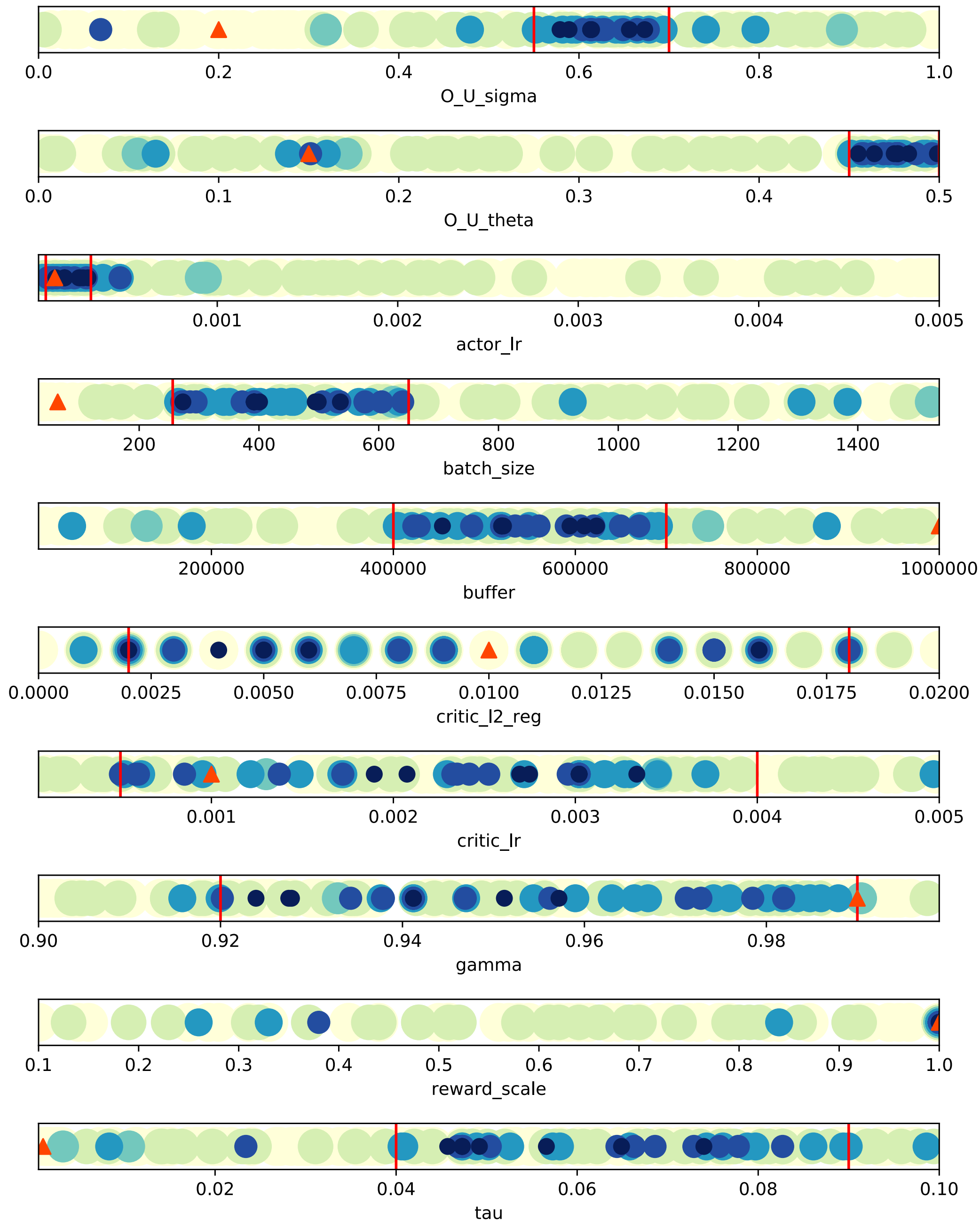


Figure 5.4: DDPG hyperparameter search: an overview of the results of the complete hyperparameter search. The markers of the hyperparameter values are based upon the performance of the corresponding agent. The large yellow markers represent the worst results, the smallest dark blue markers represent the best results. The red vertical lines are the boundaries specified in table 5.2. The bright red triangular markers represent the hyperparameter settings suggested in Lillicrap et al. [20]. For many of these parameters, we see large differences between these suggested values and the most interesting ranges for this environment. Note that the discrete appearance of the results for the L2 variable is because the values were rounded on the fourth decimal.

It is not surprising that the starting mean of the noise generation process is also larger, as stronger exploration is required in the first episodes. Because the best performing setting is found at the edge of the considered range, it might be interesting to extend the considered range. The actor learning rated suggested in the original paper [20] is strongly confirmed here. This seems one of the most critical hyperparameters, as no decent results were found using a deviant setting. We find best results when using a larger batch size, this is not very surprising as larger batches lead to better estimates for the gradient when updating the underlying models. The problem with larger batch sizes is that the environment is often computationally too demanding for larger batch sizes. We achieve better results with a smaller buffer size than suggested. For this environment, an increased buffer size does not seem to offer an advantage. The setting of the L2 regularisation for the Q-network does not seem decisive for this environment, setups with good performance are found over the whole range of this parameter. Note that the values tested for the L2 regularisation weight seem step-wise because they were rounded on the fourth decimal. The best performance was found using learning rates for the Q-network around 2 to 3 times as much as the learning rate suggested in original paper [20]. The best results for this environment were found using a slightly smaller discount rate to calculate the Q-values with the Bellman equation. This is not surprising given the very high frequency of the reward signal compared to other environments. We did not find any evidence of an advantage linked to using a rescaled reward in this environment. This might be more interesting in cases with a sparser reward space. Agents seem to perform better in this environment when the target networks follow their counterparts more closely than suggested by the original DDPG paper [20].

Finally, we wish to remark the best-performing agents are all found within the second part of this hyperparameter search. It is possible that the importance of one or a few hyperparameters causes other parameters to seemingly only perform well in this second range, while in reality, they can lead to good performance over the whole range. We conclude that we can confirm good results for these hyperparameter ranges, but can not exclude good performance for different hyperparameter settings.

5.2 Results and discussion

The summarised results for the continuous approach to the base case can be found in table 5.3. The demand scenarios are the same as for the discrete base case. As in the previous sections,

Demand	Agent	Reward	Revenue	Prod.	Hold	Trans.	Setup	Opp.
Stationary mean	DDPG	27.79	29.59	0.99	0.23	0.54	0.37	0.04
and variance	(Q,R)	25.35	27.28	0.91	0.29	0.45	0.19	0.27
Non-stationary	DDPG	27.33	29.17	0.97	0.25	0.54	0.38	0.08
variance	(Q,R)	24.66	26.64	0.89	0.32	0.44	0.19	0.34
Non-stationary	DDPG	27.04	28.94	0.97	0.28	0.54	0.39	0.11
mean	(Q,R)	24.51	26.55	0.89	0.36	0.44	0.20	0.34
Non-stationary	DDPG	26.85	28.74	0.96	0.27	0.54	0.38	0.13
mean and variance	(Q,R)	24.33	26.37	0.88	0.35	0.44	0.20	0.36

Table 5.3: Results for the base case using a continuous action space. The DDPG agent outperforms the (Q,R)-agent across all demand scenarios. The DDPG agent performs better than the NFQ agent both in general and in product availability. In contrast with the NFQ agent, the DDPG agent is able to reduce its holding costs to a level considerably lower than the (Q,R)-agent inventory costs.

the values for the reward and broken down components of the reward are expressed per unit demand to correct for fluctuations in the total amount of demand across episodes. The values are averages found over 1000 tested episodes.

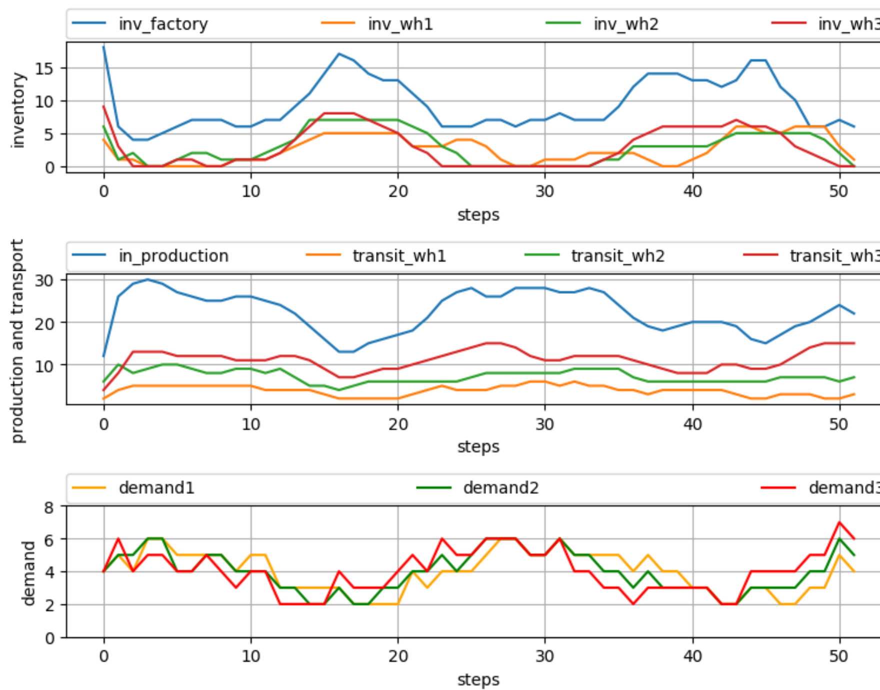
NOTE: During the hyperparameter search, the DDPG agents were not able to outperform the (Q,R)-heuristic method. Here we see that a DDPG agent trained for 6 million time steps is able to outperform the (Q,R)-agent across all demand scenarios with a considerable difference. This is due to a difference in the environment. During the hyperparameter search the environment had the same dimensions (1 factory, 3 shops, 1 product), but with lower demand. With the demand in the range of 1 or 2 products per time step, the actions ranges were also a lot smaller. This practice of using a more constrained action space was adopted from training the NFQ agents, to initially simplify the problem to be able to learn faster. However, while for discrete problems using a heavily constrained action space makes the problem easier, the opposite is true for agents using a continuous action space. This is due to the action space only being explicitly constrained for the environment and not explicitly for the agent, resulting in the agent having to learn that only actions in a very small range are purposeful.

From the results in table 5.3 we learn that with DDPG, also in the continuous case the RL agent is able to achieve a distinctly better reward than the baseline method. In contrast with the discrete case, the DDPG agent not only is able to create a much higher product availability but also manages to cut holding costs. The DDPG agent manages to incur close to 30% less holding costs while ensuring 8%-points higher product availability than the (Q,R)-agent.

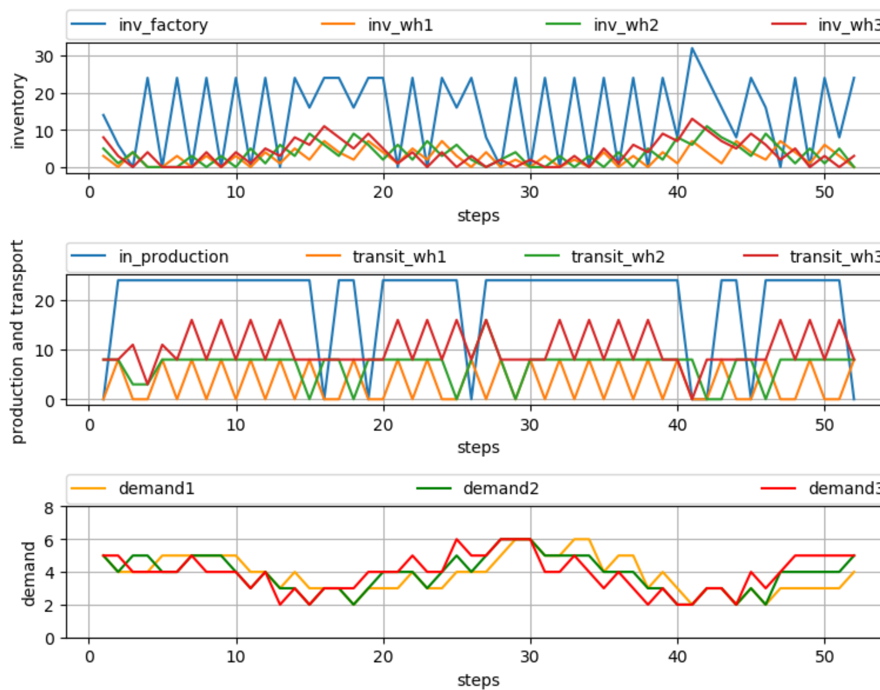
We note that the (Q,R)-agents in the discrete and continuous cases attain very similar results, as expected. From this, we conclude that the continuous (Q,R)-agent experiences little benefit from the increased action space. When the inventory in the shops drops below the reorder point, but insufficient inventory is available in the factory to supply all demanding shops, the continuous agent is able to evenly distribute all remaining factory inventory. The discrete heuristic can only distribute half of the EOQ amount to all demanding shops or not act until the factory inventory has increased.

When studying the other components of the reward function we see that the setup costs for the continuous RL agent are larger than those incurred by the (Q,R)-agent and the discrete RL agent. This further increase in the setup costs is an indication that the agent further trades off setup costs for decreased holding costs and possibly increased revenue. In practice, this corresponds to producing and transporting in smaller batches. The same is true in a lesser degree for the transportation costs, which are calculated per truck instead of per product. The transportation costs are also higher for the continuous RL agent than both the heuristic method and the discrete RL agent. The production cost for both the RL agent and the (Q,R)-agent are similar to those of their discrete counterparts. Note that a production cost of 1 corresponds to producing 1 product for every product demanded, as the unit production cost is 1. The fact that the (Q,R)-agent's production fluctuates around 0.90, hints at the possibility of increasing the reward by increasing products held. Again this will be a trade-off between the holding cost, and the production cost and the revenue, based on the reorder point.

Figures 5.5a and 5.5b offer an insight into the behaviour learned by the continuous RL agent and the characteristic inventory patterns of the (Q,R)-agent. We notice that the differences we saw between the (Q,R)-agent and the RL agent in the discrete case, have become more elaborate in the continuous case. Both the inventory levels and the stream of products in production and transportation behave more constant. This is possible because of the more fine-grained ability to select actions.



(a) DDPG agent



(b) (Q,R)-agent

Figure 5.5: Inventory evolution during one episode for the base case, using a continuous action space. The inventory build up is more outspoken for the RL agent, but the levels are lower overall compared to the (Q,R)-agent. The RL agent acts in a reactive way, very closely following the demand level with the production and transportation levels, seen in the middle panel. The RL agents inventory, production and transportation levels evolve smoother compared to all other agents.

For the continuous RL agent the differences in inventory level between periods of low and high demand. Note that the real inventory level is distinctly lower than those of the (Q,R)-agent and the NFQ agent. The points where the inventory reaches its highest level for the DDPG agent, during low-demand, are comparable to the lowest inventory levels of the NFQ agent, which are during high-demand.

The response of the RL agent to the varying demand can be seen in the shape of the curves for products in production and transportation. These all closely mimic the demand pattern. We wish to stress that the current time step is not a part of the state space. This means that the time step cannot influence the agent's behaviour. The decisions are solely based on the demand of the past three time steps, the current inventory level and the number of products in production and transportation.

Finally, corresponding with the results in table 5.3, we see that the internal behaviour of the continuous and discrete (Q,R)-agents is very similar, as expected. The less restricted action space is only taken advantage of in a handful of cases. These situations can be recognised as the irregularities in the sawtooth pattern of the products in transportation. Again, the production action always produces the EOQ amount because the production is not limited by the availability of other products.

Chapter 6

Conclusion

The problem of inventory optimisation can be described as the optimisation of company profit in the face of uncertain demand by determining when, where and what to produce and transport. The objective of this work was to further research how this problem can be dealt with in supply chains, and by extension companies in general, using reinforcement learning. To this end we created a simulation program, imitating the inventory management aspect of a supply chain. This allowed us to develop, test and compare the results of different RL agents and a benchmark representing current inventory management practice.

As mentioned in the introduction, the current methods to handle inventory management require their behaviour to be determined upfront, e.g. the (Q,R)-agent is based on the order size and the reorder point. From the experiments conducted in this work, we learn that RL agents can outperform these methods by implicitly changing these variables based on the perceived state. The DDPG agent demonstrated the behaviour of adapting its actions to the situation. In episodes with little variance, it held less safety stock than in episodes with high demand uncertainty.

In this study, we notice that the RL agents' interaction with the problem environment took a more reactive than proactive form. We expected the agent might be able to anticipate future demand, based on patterns seen before. Instead, we see that the agent does not predict the next action, but closely monitors the evolution of the state and responds to this. It is possible that the agent does not have enough information about the past evolution of the demand to chose actions anticipating future demand.

In the data from the conducted experiments, we did not find the expected signs of an integrated approach across locations in the actions of the RL agents. The setup cost could be minimised by taking transportation actions for different locations simultaneously. We expected the RL agent to exploit this, but we see that the RL agents incur higher setup costs than the (Q,R)-agents. We assume this is because the possible advantage of minimising the setup costs does not weigh up against the increased holding costs. There are other ways of acting in an integrated way, but these are more difficult to recognise from the data.

The scalability of an RL inventory optimisation program to larger problem sizes is one of the main concerns for this application. To our knowledge, the inventory management problems solved by RL agents in this work are the largest for which this has been done in the existing literature. We confirm that the possibilities for algorithms using discrete action spaces are limited in terms of problem size for inventory management cases. By solving a reasonable inventory management problem using a sample efficient, function approximator based algorithm made for continuous action spaces such as DDPG, we take a first step in the direction of tackling larger inventory management problems using RL. The question remains: how large action spaces and problems, in general, can become for DDPG to remain able to sufficiently explore the solution space. Because large amounts of training data will be needed to achieve this, it is an advantage of the inventory optimisation problem in this aspect that it can be simulated in a relatively cost-efficient way, as we have done here.

In the process of training a DDPG agent for this problem, we find evidence that corroborates the general notion that the DDPG algorithm is difficult to tune for hyperparameters. By studying the performance and evolution of 200 DDPG agents characterised by 193 different hyperparameter settings, we determine a set of interesting ranges for each hyperparameter, for inventory management problems.

Overall we conclude that the RL approaches to solving inventory management problems used in this work can compete with the existing methods for the considered setups. When comparing between the two tested RL methods, we find that the approach using DDPG, with the continuous action space, has a distinct advantage over using the NFQ algorithm. The RL agents outperform the heuristic method in such a manner that users could deem it worthwhile to give up some part of the control. We do however want to stress, that giving up control in this domain is not

a requirement for the use of RL. The actions chosen by the RL agent could serve as a guideline that could be adopted or discarded by the responsible employees.

Future Research

One of the most challenging next tasks for this field of research is to study the behaviour of the tested RL approaches on larger problems. With increasing problem size, we expect the required network size to increase as well. Which will make training more complex. Future research might look into the possibility of using more advanced network shapes to improve network performance. One way of doing this is by adding convolutional layers to the network. It might be possible to take advantage of the structured character of the data. We expect patterns to be similar for different product-location combinations. To take advantage of this we suggest using a filter on the data that takes in the information of all state components for one product-location at a time. There are two ways to achieve this. The first is using a 2D-convolution layer. The data is structured across two dimensions, using only one channel. One axis has all product-location combinations, the other all state components. The filter slides over the matrix in the direction of the product-locations, covering all state components for this product location. The second way to do this is by using a 3D convolution layer. The data is structured as a tensor with products on the first axis, locations on the second and the state components on the third, again with only one channel. The filter moving over the data takes into account one product-location combination in this plane of the tensor and all state combinations.

Comparing the performance of other RL algorithms against the results found here is another interesting research topic. We suggest using the current state-of-the-art policy gradient methods. This might be a good fit for this problem because they are regarded as less difficult to tune in terms of hyperparameters. The fact that they are less sample efficient is not problematic for this environment because of its cost-efficient simulation.

While in this paper we were not able to confirm the integrated nature of the RL agents' behaviour, we still believe this to be the case. Future works could attempt to prove this. This could be done by setting the environment parameters in such a way that a cost encouraging the integrated behaviour, such as the setup cost in this study, is more important. Another approach is to compare two RL approaches, where in the first case the decisions for every product-location combination are determined by an independent agent. In the second case, decisions are taken

by a central agent as in this work. The difference between these two approaches should indicate the advantage of centralised decision making.

In this study we found that the RL agent behaved more in a reactive than a proactive way. To enable the agent to develop a expectation for future demand, it might be necessary to give the agent access to a larger part of the past demand. It would be interesting to study the information used in demand forecast literature and attempt to include similar information in the state space of this problem.

Finally, the robustness of the interesting ranges for hyperparameters found in this work can be confirmed by future research. A comparison could be made between agents sampled from inside and outside the found ranges, studying the effect of different problem variations on the agents' performance. Possible variations could be changing the problem dimensions, such as the number of factories, shops or products. Another way to challenge the robustness of these parameter settings is by testing different demand specifications.

Bibliography

[1] Hisham M Abdelsalam and Magy M Elassal. "Joint economic lot sizing problem for a threefffdfffdfffdLayer supply chain with stochastic demand". In: *International Journal of Production Economics* 155 (2014), pp. 272–283.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[3] Marc G Bellemare et al. "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[4] Yoshua Bengio et al. "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 41–48.

[5] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[6] Greg Brockman et al. "OpenAI Gym". In: *arXiv:1606.01540 [cs]* (June 2016). arXiv: 1606.01540 [cs].

[7] Leopoldo Eduardo Cárdenas-Barrón, Kun-Jen Chung, and Gerardo Treviño-Garza. *Celebrating a century of the economic order quantity model in honor of Ford Whitman Harris*. 2014.

[8] Peter Dayan and CJCH Watkins. "Q-learning". In: *Machine learning* 8.3 (1992), pp. 279–292.

[9] Bram Desmet, El Houssaine Aghezzaf, and Hendrik Vanmaele. "A normal approximation model for safety stock optimization in a two-echelon distribution system". In: *Journal of the Operational Research Society* 61.1 (2010), pp. 156–163.

[10] Bram Desmet, El-Houssaine Aghezzaf, and Hendrik Vanmaele. “Safety stock optimisation in two-echelon assembly systems: normal approximation models”. In: *International Journal of Production Research* 48.19 (2010), pp. 5767–5781.

[11] Ford W Harris. “How many parts to make at once”. In: (1913).

[12] Matthew Hausknecht et al. “A neuroevolution approach to general atari game playing”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (2014), pp. 355–366.

[13] Robert Hecht-Nielsen. “Theory of the backpropagation neural network”. In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[14] Peter Henderson et al. “Deep Reinforcement Learning That Matters”. In: *arXiv:1709.06560 [cs, stat]* (Sept. 2017). arXiv: `1709.06560 [cs, stat]`.

[15] Henry. *Solving Continuous Control Environment Using Deep Deterministic Policy Gradient (DDPG) Agent*. en. https://medium.com/@kinwo/solving-continuous-control-environment-using-deep-deterministic-policy-gradient-ddpg-agent-5e94f82f366d. Nov. 2018.

[16] Ashley Hill et al. *Stable Baselines*. `https://github.com/hill-a/stable-baselines`. 2018.

[17] Yuenan Hou et al. “A novel DDPG method with prioritized experience replay”. In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (2017), pp. 316–321.

[18] Lukas Kemmer et al. “Reinforcement learning for supply chain optimization”. In: ().

[19] Roman Liessner et al. “Hyperparameter Optimization for Deep Reinforcement Learning in Vehicle Energy Management”. In: *ICAART*. 2019.

[20] Timothy P. Lillicrap et al. “Continuous Control with Deep Reinforcement Learning”. In: *arXiv:1509.02971 [cs, stat]* (Sept. 2015). arXiv: `1509.02971 [cs, stat]`.

[21] Long-Ji Lin. “Self-improving reactive agents based on reinforcement learning, planning and teaching”. In: *Machine learning* 8.3-4 (1992), pp. 293–321.

[22] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *arXiv:1312.5602 [cs]* (Dec. 2013). arXiv: `1312.5602 [cs]`.

[23] Steven Nahmias and Tava Lennon Olsen. *Production and operations analysis*. Waveland Press, 2015.

[24] Matthias Plappert et al. “Parameter Space Noise for Exploration”. In: *arXiv:1706.01905 [cs, stat]* (June 2017). arXiv: `1706.01905 [cs, stat]`.

[25] Martin Riedmiller. “Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method”. en. In: *Machine Learning: ECML 2005*. Ed. by David Hutchison et al. Vol. 3720. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 317–328. ISBN: 978-3-540-29243-2 978-3-540-31692-3. DOI: `10.1007/11564096_32`.

[26] Martin Riedmiller and Heinrich Braun. “A direct adaptive method for faster backpropagation learning: The RPROP algorithm”. In: *Proceedings of the IEEE international conference on neural networks*. Vol. 1993. San Francisco. 1993, pp. 586–591.

[27] Tom Schaul et al. “Prioritized Experience Replay”. In: *arXiv:1511.05952 [cs]* (Nov. 2015). arXiv: `1511.05952 [cs]`.

[28] Joaquín Sicilia et al. “An inventory model for deteriorating items with shortages and time-varying demand”. In: *International Journal of Production Economics* 155 (2014), pp. 155–162.

[29] Edward A Silver. “Operations research in inventory management: A review and critique”. In: *Operations Research* 29.4 (1981), pp. 628–645.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. en. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press, 1998. ISBN: 978-0-262-19398-6.

[31] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[32] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[33] George E Uhlenbeck and Leonard S Ornstein. “On the theory of the Brownian motion”. In: *Physical review* 36.5 (1930), p. 823.

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

[35] Matthew Waller, Brent D Williams, and Travis Tokar. “A review of inventory management research in major logistics journals”. In: *The International Journal of Logistics Management* (2008).

Appendix A

Complete results tables

Demand	Agent	Value	Reward	Revenue	Prod.	Hold	Trans.	Setup	Opp.
Stationary mean and variance	NFQ	Mean	26.76	28.95	0.99	0.32	0.48	0.29	0.11
		Std. dev	0.34	0.34	0.02	0.02	0.01	0.01	0.03
	(Q,R)	Mean	25.23	27.38	0.92	0.31	0.46	0.19	0.26
		Std. dev	0.42	0.41	0.02	0.02	0.01	0.01	0.04
Non-stationary variance	NFQ	Mean	26.82	29.13	1.01	0.40	0.49	0.32	0.09
		Std. dev	0.48	0.46	0.02	0.03	0.01	0.01	0.05
	(Q,R)	Mean	24.57	26.77	0.90	0.34	0.45	0.19	0.32
		Std. dev	0.89	0.82	0.03	0.03	0.01	0.01	0.08
Non-stationary mean	NFQ	Mean	25.96	28.30	0.98	0.40	0.48	0.32	0.17
		Std. dev	0.79	0.88	0.03	0.14	0.02	0.03	0.09
	(Q,R)	Mean	24.45	26.72	0.91	0.37	0.45	0.20	0.33
		Std. dev	1.43	1.50	0.06	0.14	0.03	0.03	0.15
Non-stationary mean and variance	NFQ	Mean	26.23	28.63	1.00	0.47	0.48	0.32	0.14
		Std. dev	0.51	0.57	0.02	0.13	0.01	0.02	0.06
	(Q,R)	Mean	24.15	26.41	0.90	0.36	0.44	0.19	0.36
		Std. dev	1.54	1.60	0.06	0.13	0.03	0.02	0.16

Table A.1: Base case using discrete action space: full results

Demand	Agent	Value	Reward	Revenue	Prod.	Hold	Trans.	Setup	Opp.
Stationary mean and variance	NFQ	Mean	15.43	21.98	4.03	0.54	1.01	0.60	0.36
		Std. dev	0.18	0.19	0.04	0.03	0.01	0.01	0.04
	(Q,R)	Mean	11.47	17.41	3.22	0.46	0.80	0.30	1.14
		Std. dev	0.14	0.12	0.06	0.02	0.01	0.01	0.03

Table A.2: Expanded case using a discrete action space: complete results

Demand	Agent	Value	Reward	Revenue	Prod.	Hold	Trans.	Setup	Opp.
Stationary mean and variance	DDPG	Mean	27.79	29.59	0.99	0.23	0.54	0.37	0.04
		Std. dev	0.16	0.16	0.01	0.02	0.01	0.01	0.02
	(Q,R)	Mean	25.35	27.28	0.91	0.29	0.45	0.19	0.27
		Std. dev	0.40	0.39	0.02	0.02	0.01	0.01	0.04
Non-stationary variance	DDPG	Mean	27.33	29.17	0.97	0.25	0.54	0.38	0.08
		Std. dev	0.56	0.52	0.02	0.03	0.01	0.01	0.05
	(Q,R)	Mean	24.66	26.64	0.89	0.32	0.44	0.19	0.34
		Std. dev	0.82	0.76	0.03	0.03	0.01	0.01	0.08
Non-stationary mean	DDPG	Mean	27.04	28.94	0.97	0.28	0.54	0.39	0.11
		Std. dev	0.93	1.01	0.04	0.15	0.03	0.06	0.10
	(Q,R)	Mean	24.51	26.55	0.89	0.36	0.44	0.20	0.34
		Std. dev	1.41	1.47	0.06	0.14	0.03	0.03	0.15
Non-stationary mean and variance	DDPG	Mean	26.85	28.74	0.96	0.27	0.54	0.38	0.13
		Std. dev	1.10	1.15	0.04	0.14	0.03	0.06	0.11
	(Q,R)	Mean	24.33	26.37	0.88	0.35	0.44	0.20	0.36
		Std. dev	1.55	1.58	0.06	0.14	0.03	0.03	0.16

Table A.3: Base case using a continuous action space: full results