Spare Parts Inventory Control Using Reinforcement Learning

Simons, A.R.

Award date:
2021

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Stochastic Operations Research Group

Spare Parts Inventory Control Using Reinforcement Learning

Master Thesis

A.R. Simons

Supervisors:
dr. S. Kapodistria (TU/e)
prof. dr. ir. S.C. Borst (TU/e)
dr. ir. B.W.J. Mathijsen (CQM)
dr. ir. J. van Doremalen (CQM)

Eindhoven, August 2021

Abstract

Objective: The inventory control problem can mathematically be modelled as a Markov decision process. To solve a Markov decision process to optimality explicit knowledge of the model is needed, which might not be available in practice. Another limitation is that the problem becomes intractable for growing state and/or action space. A potential alternative that could overcome these issues is reinforcement learning. The objective is to gain a better understanding of reinforcement learning, how it works and if it is applicable to spare parts inventory control.

Methodology: To this end two reinforcement learning methods were investigated: REINFORCE and Q-learning. For Q-learning three versions with different levels of complexity are considered: tabular, double, and deep Q-learning. In order to have a baseline for performance, a stylized model is formulated as a Markov decision process and solved to optimality. The applicability was tested on a case at Philips Healthcare which involved data that needed to be analyzed.

Findings: REINFORCE tends to converge to a local optimum and finds sub-optimal policies with a small deviation from the optimal policy, but it is the most consistent method for the Philips case. Tabular Q-learning is guaranteed to converge and is the best performing method for small instances, but needs much time to converge for large instances. Double Q-learning does not have an advantage over tabular Q-learning in the Philips case. DQN is the least consistent method, but might work for large instances.

Managerial Insights: Reinforcement learning methods do not come with rigorous performance guarantees in general and their performance highly depends on the problem. For single-item single-echelon spare parts inventory models REINFORCE and tabular Q-learning are useful. Which one is preferred over the other depends on the need for an optimal solution versus the time that one has for learning.

Link to original code: *https://github.com/ankesimons/SpareParts-RL*

Keywords: *Reinforcement learning, REINFORCE, Q-learning, Spare parts, Inventory control, Markov decision process, Infinite-horizon, Discounted rewards.*

Preface

This thesis has been written to fulfill the graduation requirements for the degree of Master of Science in Industrial & Applied Mathematics, Eindhoven University of Technology.

I would like to thank my academic supervisors Stella Kapodistria and Sem Borst, and my industrial supervisors Britt Mathijsen and Jan van Doremalen, for their guidance, critical feedback and support during the process of writing this thesis. I wish to thank CQM, for giving me the opportunity to write my thesis at a company, and my colleagues at CQM for their interest in this project.

I also wish to thank my friends and family, who have always supported me, even if they did not understand a thing of what I was doing. In particular I would like to thank my parents for their counsel and for keeping me motivated when the process was not going as smoothly as I had hoped for.

This thesis marks the end of eight amazing years of studying. The road to my graduation was not always easy, but I have learned a lot during my studies and I am very proud of what I have achieved.

Anke Simons

Contents

Contents iii

List of Figures vi

List of Tables viii

List of Symbols ix

1 Introduction 1

1.1 Spare Parts Management at Philips Healthcare 2

1.2 Outline of the Thesis 4

2 Literature Study 5

2.1 Stochastic Decision Making 5

2.2 Reinforcement Learning 6

2.2.1 Policy and Value Based Learning 8

2.3 Deep Reinforcement Learning 8

2.3.1 Function Approximation 8

2.4 Spare Parts Networks 9

2.5 Reinforcement Learning and Spare Part Networks 11

3 A Stylized Model 12

3.1 Finite Markov Decision Process 13

3.1.1 The Optimality Criterion 15

3.1.2 Discount and Average Optimal 17

3.2 Dynamic Programming 17

3.2.1 Policy Iteration 17

3.2.2 Value Iteration 18

3.3 Simulation Environment 18

3.4 Example for Performance 19

4 REINFORCE **20**
4.1 Mathematical Formulation . . . 21
4.2 Challenges . . . 22
4.2.1 Policy Parameterization Example . . . 23
4.2.2 Discounted Sum of Rewards . . . 24
4.2.3 Non-convex Loss Function . . . 26
4.3 Example . . . 28
4.4 Concluding Remarks . . . 29

5 Q-learning **30**
5.1 Mathematical Formulation . . . 30
5.2 Tabular Q-learning . . . 31
5.2.1 Convergence Properties . . . 31
5.2.2 Implementation . . . 32
5.2.3 Example . . . 33
5.3 Double Q-learning . . . 33
5.3.1 Implementation . . . 34
5.3.2 Example . . . 34
5.4 Deep Q-learning . . . 35
5.4.1 Implementation . . . 36
5.4.2 Example . . . 37
5.5 Concluding Remarks . . . 38

6 Data Analysis **40**
6.1 Demand Analysis . . . 41
6.1.1 Distribution Fitting . . . 44
6.2 Cost Analysis . . . 48
6.2.1 Emergency Cost Tuning . . . 48

7 Results Stylized Model **50**
7.1 SKU 1 . . . 50
7.2 SKU 2 . . . 52
7.3 SKU 3 . . . 53
7.4 Concluding Remarks . . . 55

8 A Model Extension **56**
8.1 Finite Markov Decision Process . . . 57
8.2 Validation . . . 58
8.3 Concluding Remarks . . . 61

9 Conclusions and Future Research **62**
9.1 Findings . . . 62
9.1.1 Research and Performance RL Methods . . . 62
9.1.2 Model Extension . . . 63
9.1.3 Conclusion . . . 63
9.2 Limitations and Future Research . . . 63
9.2.1 Reinforcement Learning Methods . . . 63
9.2.2 Data Analysis . . . 64
9.2.3 Model Complexity . . . 64

Bibliography **65**

Appendix **69**

A Bounded Rewards **69**

B Non-convexity of the REINFORCE Loss Function **70**

C Dynamic Programming **72**
C.1 Policy Iteration . . . 72
C.2 Value Iteration . . . 73

D Regression Transportation Cost **74**

E Results Stylized Model Extra Figures SKU 1 **75**

List of Figures

1.1 Flow chart TTSU 3
1.2 Service part supply chain logistic flow 3

2.1 The agent-environment interaction in a Markov decision process. 7

3.1 Stylized model flow chart. 12
3.2 Events in the stylized model. 13

4.1 Overview structure REINFORCE. 21
4.2 Example neural network as policy estimator 23
4.3 Visualization convergence REINFORCE, stylized model. 29

5.1 Visualization convergence Q-learning, stylized model. 33
5.2 Visualizations convergence double Q-learning, stylized model. 35
5.3 Visualization learning DQN, stylized model 38
5.4 Learned Q-values example stylized model. 39

6.1 Scatterplots prices versus order volumes of SKU's in the SPS data set. 41
6.2 Histograms order volume frequency per SKU. 42
6.3 Order volume boxplots per SKU. 43
6.4 QQ-plots demand distribution SKU 1. 45
6.5 QQ-plots demand distribution SKU 2. 46
6.6 QQ-plots demand distribution SKU 3. 47
6.7 Histograms order volume frequency per SKU, with fitted demand distribution. . 47

7.1 Learned policies, SKU 1 state 0-25. 51
7.2 Results SKU 1 51
7.3 Learned policies SKU 2. 52
7.4 Results SKU 2. 53
7.5 Learned policies SKU 3. 54

7.6 Results SKU 3. 54

8.1 Model extension flow chart. 56
8.2 Q-learning, model extension validation case 3. 60
8.3 Q-learning, model extension validation case 4. 61

E.1 Learned Q-values for 100.000 simulated trajectories for SKU 1. 75
E.2 Learned policies SKU 1. 76

List of Tables

3.1 Parameter settings example stylized model. 19

4.1 Simulation results bias correction discounted sum of rewards. 26

5.1 Overview Q-learning results for stylized model example. 38

6.1 Description of the columns of interest in the SPS dataset. 40
6.2 Price, weight, orders, order volume per SKU. 42
6.3 Daily order volume per SKU. 42
6.4 Order volume summary statistics per SKU. 44
6.5 Parameter estimation for several distributions, SKU 1. 45
6.6 Parameter estimation for several distributions, SKU 2. 46
6.7 Optimal policy robustness under two demand distributions, SKU 2. 46
6.8 Parameter estimation for several distributions, SKU 3. 47
6.9 Cost and capacity parameters per SKU. 48
6.10 Part availability rates per SKU. 48

7.1 Results SKU 1. 52
7.2 Results SKU 2. 53
7.3 Results SKU 3. 54

8.1 Parameter settings validation cases model extension. 59
8.2 Optimal policy and values, model extension validation case 3. 60
8.3 Policies, model extension validation case 4. 61

B.1 Study non-convexity loss function REINFORCE, example 1. 70
B.2 Study non-convexity loss function REINFORCE, example 2. 71

D.1 Total weight and cost to send items with the UPS service. 74
D.2 Output regression analysis transportation cost. 74

List of Symbols

$\mathcal{T}$	Decision epochs
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
S_t	State at time t
A_t, a_t	Action taken at time t
D, D_t	Customer demand
I_t	Inventory level at time t
$\widehat{D}$	Excess demand
$\widehat{I}$	Inventory surplus
R	Direct one-step reward
$r(s, a)$	Expected one-step reward when taking action a in state s
s_{max}	Inventory capacity
a_{max}	Order capacity
h	Holding cost per part, per time unit
e	Emergency cost per part
τ_v	Variable transportation cost
τ_f	Fixed transportation cost
ρ	Penalty cost per part for the model extension
π	Policy
π^*	Optimal policy
π_θ	Policy parameterization in terms of $\boldsymbol{\theta}$
$\boldsymbol{\theta}$	Weights of a neural network
γ	Discounting factor

G_t^π	Discounted sum of rewards under policy π from time-step t onward
$G_{t,T}^\pi$	Finite horizon estimator for G_t^π
T	Total trajectory length
V^π	State value function under policy π, i.e. expected value of G_t^π given S_t
V^*	Optimal state value function
Q^π	State-action value function under policy π, i.e. expected value of G_t^π given S_t and a_t
Q^*	Optimal state-action value function
$Q_{\boldsymbol{\theta}}$	State-action value function approximation, a parameterization in terms of $\boldsymbol{\theta}$
$\mathcal{L}$	Loss function
α, α_n	Learning rate, learning rate in iteration n

Chapter 1

Introduction

Spare parts, parts that can replace broken parts in a machine, are important for the maintenance of machines and equipment. Spare parts management is a widely studied area in operations research. Spare parts inventory control is a subarea in which the main challenge is to determine optimal replenishment strategies. Mathematically, the inventory control problem can be modelled as a Markov decision process (MDP), which is a powerful framework for (stochastic) decision making.

A classical method to solve an MDP is dynamic programming. Under suitable conditions this method finds the numerical (optimal) solution to the MDP in a finite number of iterations. This solution consists of two elements: a policy, indicating what the decision maker should do in each situation in order to receive maximized rewards in the long run, and the value function, representing the expected rewards in the long run from each state onward.

One of the limitations of dynamic programming is that it suffers from the curse of dimensionality; it becomes intractable for growing state and/or action space. Furthermore an MDP relies on an explicit model and parameter knowledge which may not be available in practice. A potential alternative that could overcome these problems is reinforcement learning.

Reinforcement learning (RL) is a method in which a (virtual) decision maker interacts with a (virtual) environment and learns from its own experience based on earned rewards. Contrary to dynamic programming, RL does not require the decision maker to have knowledge of the underlying model dynamics in terms of the transition probabilities and reward structure, and promises to be applicable to situations with a large number of states and actions as well. RL methods can either learn the policy directly, or can derive the policy from learned values.

The main goal of this thesis is to gain a better understanding of reinforcement learning, how it works and if it is applicable to spare parts inventory control. For this purpose two RL approaches were investigated; REINFORCE and Q-learning. The distinction between these two is that REINFORCE directly learns a policy, whereas Q-learning learns the value function.

As a starting point, we consider a simplified single-echelon single-item (spare parts) inventory model, referred to as the stylized model, for which an MDP formulation will be provided. An example will be provided for which the optimal solution is determined. This solution will be used as a baseline for the other methods discussed in this thesis.

In REINFORCE the policy will be represented by a neural network, and optimized via a gradient descent method. Due to non-convexity of the function that is being optimized in this method, REINFORCE tends to converge to a local optimum, instead of the global optimum.

Therefore we have investigated a second RL method: Q-learning.

Q-learning is a method that learns the value function directly. From the learned values the policy can be derived. There exist several versions of Q-learning. The simplest form of Q-learning is tabular Q-learning, for which convergence is guaranteed under suitable conditions. A slightly more complex form is double Q-learning. This method should overcome what is known as the maximization bias for tabular Q-learning and provide better estimates of the value function. A third form is deep Q-learning, this method can handle models with a larger state and action space compared to tabular or double Q-learning, but does not have convergence guarantees.

After gathering more insight into these RL methods, we investigate the applicability to a case at Philips Healthcare. To this end we performed data analysis to specify demand distributions and associated costs for three specific spare parts. In Section 1.1 below, we provide more details on the Philips Healthcare case.

For the Philips case we will compare the results for the aforementioned RL methods to the optimal solutions, based on both the policy and the value function. Tabular Q-learning is the only method that resulted in the optimal policy for two of the three spare parts in the Philips case, but the convergence is slow for problems with a large state space. REINFORCE results in sub-optimal policies, with a small deviation from the optimal policy, but is the most consistent method for the Philips case. DQN is the least consistent method. We cannot give a conclusive explanation for the performance of DQN, as it is not as thoroughly investigated as the REINFORCE, tabular and double Q-learning, but the results show that it could possibly be useful for large instances.

The majority of this thesis is devoted to the stylized model. The Philips spare part supply chain is more complex than that, which motivated us to extend the stylized model. We conclude this thesis by proposing a model extension. Due to time limitations we applied only one of the RL methods to the model extension, namely tabular Q-learning because of the convergence guarantees, and validate the method using various intuitive examples.

This research was conducted at the company CQM (Consultants in Quantative Methods). The case to investigate applicability was provided by Philips Healthcare.

1.1 Spare Parts Management at Philips Healthcare

Philips is one of the largest producers of large medical systems, such as MRI scanners, which are often used for critical processes in hospitals. These systems are quite expensive, which causes hospitals to only have a limited number of these systems. Therefore it is important that these systems remain up and running. However, it is unavoidable that these systems sometimes break down. If that happens the manufacturer provides service to the customer and aims to have the system up and running again as soon as possible, taking into account the availability of the necessary spare parts and an engineer if needed.

The total time a system is down is measured by the time till system up (TTSU) as a key performance indicator (KPI), which is specifically defined as the time elapsed between a customer complaint call and closing of the case. Philips aims to limit the TTSU to 24 hours for 90% of the system-down cases on a yearly basis.

If a system is down the customer calls and a case will be created. Once the case is created, a diagnosis is to be made to identify the problem that caused the system to go down. Sometimes this can be done remotely, otherwise a service engineer has to do this on location. Once the diagnosis has been made it should be clear which spare parts are required and a service engineer

goes to the customer with the required spare parts to repair the system. This process is visualized in Figure 1.1 obtained from [22].

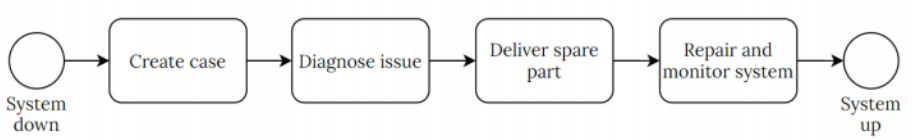


Figure 1.1: Flow chart TTSU

This research focuses on the inventory control of the spare parts, which plays an important role in the 'deliver spare part' section in the TTSU process. If there are not enough spare parts available, it takes longer to deliver a spare part, which causes the TTSU to rise.

The supply chain network for spare parts is managed by the department Service Parts Supply (SPS). The SPS department consists of three types of links: regional distribution centers (RDC), local distribution centers (LDC) and forward service locations (FSL), see Figure 1.2 obtained from [22]. The regional distribution centers are supplied by four different types of suppliers and can distribute spare parts to either the LDC's, FSL's or the customers. The LDC's can distribute to the FSLs or the customers and the FSL's can only distribute to the customers directly. From either one of these three links the spare parts can thus be distributed to the customer. This research will focus on replenishment of the warehouses in the SPS section.

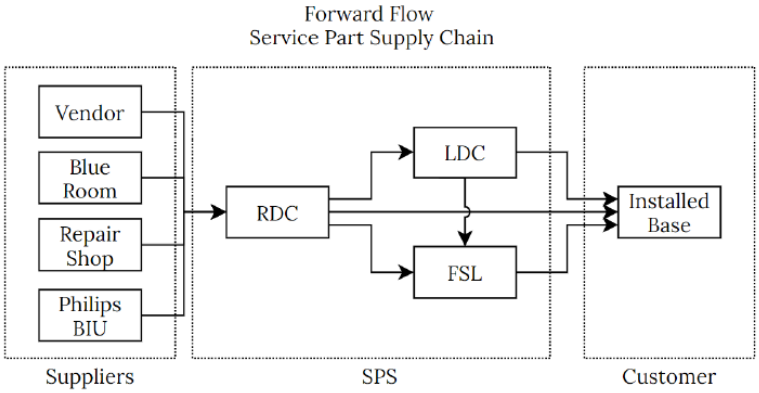


Figure 1.2: Service part supply chain logistic flow

The medical systems manufactured by Philips are complex systems with many different parts. Some spare parts are requested on a regular basis, but others are requested only on rare occasions. This causes the inventory control to be a challenging task. An extra challenge in the management of spare parts for Philips is the goal to solve 90% of all the system-down cases within 24 hours. This target is translated to a target of 95% availability of spare parts for the SPS department, to allow for variability in the other processes within the TTSU flow [22].

The main objective in the Philips case is to have a 95% spare parts availability in the warehouses. At the same time the cost should be minimized. The cost involved in managing the spare part supply chain include storage costs, warehouse handling costs and transportation costs.

1.2 Outline of the Thesis

In the next chapter we will introduce the core concepts of stochastic decision making and (deep) reinforcement learning, followed by an overview of the literature in spare parts networks. In Chapter 3 we will introduce the stylized model and provide the MDP formulation. In Chapter 4 and Chapter 5 we will discuss the investigated RL methods REINFORCE and Q-learning respectively. Chapter 6 contains the data analysis in order to connect the investigated methods to the Philips case, of which the result will be presented in Chapter 7. In Chapter 8 we propose an extension to the stylized model. In Chapter 9 we conclude this thesis and give suggestions for future research.

Chapter 2

Literature Study

This chapter contains the literature study. As mentioned in the introduction, the area of interest is spare parts networks and reinforcement learning.

First we will provide theoretical background on stochastic decision making. Then we will introduce the concept of reinforcement learning and deep reinforcement learning. Next we will elaborate on types of spare parts networks and replenishment strategies. This chapter is concluded by connecting reinforcement learning to spare parts networks.

2.1 Stochastic Decision Making

A Markov chain is stochastic model often used to describe a sequence of random events in which the probability to reach a particular subsequent state only depends on the present state. The independence of history is known as the Markov property. If a decision maker can influence the process with its decisions we refer to the process as a Markov decision process (MDP).

Markov decision processes form a powerful mathematical framework to model (stochastic) decision making. State transitions satisfy the Markov property and only take place at discrete time steps. In an MDP the state is fully observable by a decision maker, if the decision maker cannot observe an entire state we speak of a partially observable Markov decision process (POMDP) [26].

Contrary to an MDP, in a semi Markov decision process (SMDP) the state does not necessarily change at discrete time steps. State changes can take various amounts of time, which makes SMDPs appropriate for continuous-time discrete-event systems [31]. In the spare parts inventory management case we assume that the inventory level is reviewed on a daily basis. Therefore this research will focus on discrete time processes, considering MDPs in which the state changes only at discrete time steps, representing the daily reviews.

In general the objective in an MDP is to maximize the cumulative received reward in the long run. When defining the objective there are two important aspects to take into account: the time-horizon and the reward structure.

The time-horizon can either be finite or infinite, depending whether there is a specific time horizon of interest. A finite time-horizon is useful if the decision makers lifetime is limited, for example when there exists a terminal state where the process eventually will end up, like in a game where the decision maker either wins or loses. In the finite-horizon setting the total

received reward can be evaluated as long as the rewards are bounded. The total reward will be finite due to bounded rewards and bounded time.

If the decision makers lifetime is not limited the infinite-horizon setting is more applicable. In this case the time is no longer bounded and the total reward could become infinitely large. One can overcome this problem by analyzing either a discounted or average reward.

In this research the infinite-horizon, discounted reward structure is considered. The infinite-horizon, discounted reward model is mathematically more convenient than the finite-horizon model [16]. Discounting is a natural way to ensure that immediate rewards are more important than future rewards. One can think of this as inflation, where money is worth more today than tomorrow. When the discounting factor approaches 1 the discounted reward structure approximates the average reward structure [35]. Both the average and discounted reward structure have the same theoretical guarantees [34, 38].

We summarize the key elements of MDPs.

- The state space ($\mathcal{S}$), action space ($\mathcal{A}$), state transitions and transition probabilities.
- A reward signal (R), defining which actions are good or bad in an immediate sense.
- A policy (π), indicating the decision maker's behavior.
- Value function, the expected discounted sum of rewards per state starting from that state following policy π. This is a prediction of the future reward and is used to evaluate how *good* or *bad* a state is. For future reference we make a distinction between the state value (V^π), and the state-action value (Q^π). As the names suggest the state value is only depending on the state, whereas the state-action value depends on both the state and action taken from that state.
- The Bellman equation, a recursive formula that "expresses a relationship between the value of a state and the values of its successor states" [32].

When solving MDPs one tries to find a solution to the Bellmann equation. A classical method to solve MDPs is dynamic programming, policy iteration and value iteration for example. Under suitable conditions dynamic programming methods find the optimal solution to the MDP. One of the limitations of dynamic programming is that it suffers from the curse of dimensionality; it becomes intractable for growing state- and/or action-space. Furthermore, it relies on explicit knowledge of the model and parameters, which may not be available in practice. A potential alternative that could overcome these issues is reinforcement learning.

2.2 Reinforcement Learning

In practice a decision maker does not always have full knowledge of the model dynamics, which is required to solve MDPs. Instead one could have access to an oracle that mimics the model dynamics, or observations from an actual operating environment. This is where reinforcement learning could offer a solution.

Reinforcement learning (RL), also referred to as approximate dynamic programming or neuro-dynamic programming [2], is a computational approach in which an agent interacts with an environment and learns to make decisions based on a reward signal in order to achieve a long-term goal. Relating this to MDPs, the agent is the decision maker that does not have prior

knowledge of the model dynamics, but learns the dynamics from experience by exploring the state space. The environment is everything outside the agent mimicking the model dynamics. The agent's goal is to maximize the received (discounted) reward in the long run, similar to the goal of the decision maker in the MDP setting, by selecting actions such that the (discounted) sum of future rewards is maximized. The selection of actions is done according to the agent's policy.

Often a discrete event simulation is used to mimic the environment, in which only the probability distributions of the random variables affecting the process are required [7]. The simulation acts as an oracle that performs the state transitions and provides rewards as an incentive based on the actions taken by the agent. During the process the agent explores the state space and learns which actions lead to high long term rewards. Note that in RL, contrary to MDPs, the model dynamics do not necessarily need to be Markovian [42].

The interaction between agent and environment is a cyclic process, see Figure 2.1 [32]. It starts with the agent observing the current state of the environment. Based on this observation the agent decides which action to take. This action is input for the environment based on which the environment possibly makes a transition to a new state and the agent receives a reward as feedback on his decision. The new state is observed by the agent and the process repeats.

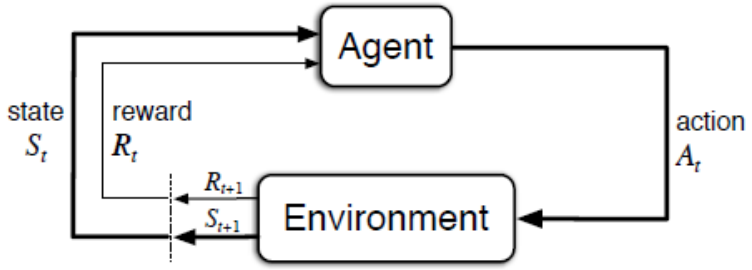


Figure 2.1: The agent-environment interaction in a Markov decision process.

Typically RL methods can be divided into model-free and model-based approaches. In model-free RL the rewards are directly optimized and the policy is learned directly [8], and in model-based RL additionally the model is employed and/or learned [1]. An explicit model of the environment, containing the transition probabilities and reward matrix, is thus not required and can, but does not necessarily need to be learned.

The agent can learn on-policy or off-policy. On-policy means learning 'on the job', the agent continuously behaves according to its own, learned, policy and at the same time updates the policy to become better. Off-policy on the other hand means learning while implementing another policy, which one can interpret as learning from an expert's policy.

One of the challenges in reinforcement learning is the exploration and exploitation trade-off. As the agents goal is to maximize the total rewards in the long run it wants to take actions that the agent knows result in high rewards, but there might be better actions not yet explored [32].

From a practical perspective we want the agent to learn the policy that optimizes the total (discounted) reward. A policy can either be learned directly or be derived from learned state-action values.

2.2.1 Policy and Value Based Learning

Policy based methods directly produce a policy, which makes the result easy to interpret. During the learning process a policy is kept in memory and adjusted based on the received rewards. REINFORCE [43] is a policy based RL method. This method uses a neural network to approximate the policy function. We will further elaborate on REINFORCE in Chapter 4.

Value based methods learn the state-action values, from which the policy directly can be derived. Q-learning [40] is a value based RL method, in which the value function is updated iteratively via a specific update rule. The value estimates are updated based on other value estimates, which is known as bootstrapping [32]. The value function can either be stored in a look-up table or approximated using a neural network. We will further elaborate on Q-learning in Chapter 5.

The usage of neural networks for approximation of the policy or value function is part of an area known as deep reinforcement learning (DRL), which we will further discuss in the next section.

2.3 Deep Reinforcement Learning

In deep reinforcement learning (DRL) deep learning is used for function approximation, to approximate either the value function or policy. Both the value function and the policy tend to exhibit a lot of structure, and could therefore be difficult and computationally burdensome to determine precisely. Approximate methods are not guaranteed to find the optimal solution, but can be effective to approximate the solutions of large problems [32].

In this section we will provide the basic concept of function approximation using a neural networks. For more details on deep learning we refer to [11].

2.3.1 Function Approximation

To approximate the policy or value function we use a neural network. The most common neural networks are feedforward neural network. This type of network consists of an input layer, one or multiple hidden layers and an output layer, and does not contain loops that feed output of the model back to itself [11]. Each layer performs a mathematical operation to the output of the previous layer. Each layer consists of a predefined number of nodes in which a transformation takes place and weights and biases are assigned. These weights and biases are the model parameters, denoted by $\boldsymbol{\theta}$, that will be updated throughout the learning process.

Mathematically the neural network architecture can be described as follows. The neural network maps the input state, represented by a vector $\mathbf{s}^0 \in \mathbb{R}^{d^0}$ to the total number of possible actions $|\mathcal{A}|$, $NN : \mathbb{R}^{d^0} \to \mathbb{R}^{|\mathcal{A}|}$. The input is propagated through L layers, and in each layer a transformation takes place. Let $\mathbf{s}^l$ be the output of the l^{th} layer, then for $l \in \{1, \ldots, L\}$

$$\mathbf{s}^l = f^l(W^l \mathbf{s}^{l-1} + \mathbf{b}^l), \tag{2.1}$$

where $W^l \in \mathbb{R}^{d^l \times d^{l-1}}$ is the matrix of weights to be assigned and $\mathbf{b}^l$ the assigned bias to each node in layer l. $f^l(\cdot)$ is the non-linear activation function. The network parameters are

$$\boldsymbol{\theta} = \{W^1, \mathbf{b}^1, \ldots, W^L, \mathbf{b}^L\}. \tag{2.2}$$

Non-linear activation layers allow to learn more complex structures than the linear layers. There are many different types of non-linear activation layers, that perform different kinds of transformations. Some layers are more suitable for specific tasks than others. The Rectified Linear Unit (ReLU), proposed in [24], is the modern default [11] and is widely used for deep learning applications [25]. The ReLU layer operates as a ramp function, a piece-wise linear function that transforms all negative output values of the previous layer to 0,

$$f(x) = \max\{0, x\}. \tag{2.3}$$

The piece-wise linearity makes this function easy to optimize with gradient-descent methods [11].

If the neural network is used to approximate the policy, the output is a probability distribution over the action space. To achieve this, the Softmax function is applied to the output. The Softmax function $g_i(\mathbf{x}) : \mathbb{R}^{|\mathcal{A}|} \to [0,1]^{|\mathcal{A}|}$ is defined as

$$g_i(\mathbf{x}) = \frac{\mathrm{e}^{x_i}}{\sum_{j \in \mathcal{A}} \mathrm{e}^{x_j}}, \tag{2.4}$$

for $i \in \mathcal{A}$ and $\mathbf{x} = (x_1, \ldots, x_{|\mathcal{A}|}) \in \mathbb{R}^{|\mathcal{A}|}$.

To measure the effectiveness of a policy or the correctness of the value function a loss function will be defined, denoted by $\mathcal{L}$, where a small loss typically indicates a good approximation. The loss function depends on the approximated policy or value function, and therefore depends on $\boldsymbol{\theta}$. What an appropriate loss function is highly depends on the problem, so we will not go into detail here. We will elaborate on the loss function for REINFORCE in Chapter 4.

The model parameters $\boldsymbol{\theta}$ are updated via a gradient descent method. Gradient descent is an optimization method for finding the (local) minimum of a function. The idea is to iteratively adjust the parameters in the direction of steepest descent as defined by the negative of the gradient. The learning rate determines the size of the step we take in the direction of steepest descent. The general update rule is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \cdot \nabla \mathcal{L}(\boldsymbol{\theta}_t), \tag{2.5}$$

where α is the learning rate and $\nabla \mathcal{L}(\boldsymbol{\theta})$ is the gradient of the function we aim to minimize with respect to the weights $\boldsymbol{\theta}$, i.e. the loss function.

Currently the gradient descent method Adaptive Moment Estimation (Adam) [19] is used very frequently in optimizing the parameters of a neural network. It is an adaptive stochastic gradient descent method, developed for the optimization of "stochastic objectives with high-dimensional parameter spaces. This method computes adaptive learning rates for different parameters from estimates of first and second moments of the gradients" [19, p. 1].

Due to non-linearity of neural networks, the loss function is usually not convex. "Stochastic gradient descent applied to non-convex loss functions has no convergence guarantee, and is sensitive to the values of the initial network parameters." [11, p. 177]. In general neural networks are sensitive to hyper-parameters such as the learning rate and batch size. The batch size indicates the size of the sample of observations that is used to estimate the gradient to perform the optimization step.

2.4 Spare Parts Networks

Spare parts are important for the maintenance and availability of equipment. Compared to 'normal' inventory control, spare parts inventory control entails a couple of challenges such as

criticallity of parts, part availability agreements, and typically a low and possibly infrequent demand. The demand for spare parts is stochastic, as one often does not know when and which spare parts are needed. Forecasting of spare parts demand is difficult due to the low and infrequent demand, and the compound nature of demand as a mixture of demand arrivals and demand sizes [4].

In spare part management the main objective is twofold. On the one hand the spare parts availability is to be maximized and on the other hand the cost is to be minimized. Therefore often a trade-off between service and costs should be made. With a higher inventory level the availability and therefore the service will be better, but also the cost will be larger.

In general there are two types of (spare part) supply chains, single-echelon and multi-echelon. In a single-echelon supply chain there is one warehouse acting as the central repository between the supplier and the customer, whereas a multi-echelon supply chain consists of multiple layers of warehouses.

Clark and Scarf [6] were the first to consider multi-echelon inventory models, where each echelon orders from its immediate upstream echelon. For this type of model optimality of the (s, S) policy, also known as the order-up-to policy, is proven under the following assumptions: demand only occurs at the lowest echelon, linear purchasing and transportation cost without setup cost between echelons, convex holding and shortage cost at each echelon, and excess demand is back-ordered. In the (s, S) policy the inventory level is periodically monitored, if the inventory level is below s the inventory is replenished up to the order-up-to level S [29].

When the back-ordering assumption is dropped and instead lost sales is considered, order-up-to policies in general perform poorly [45]. However, when a stock-out leads to a high penalty, order-up-to policies are asymptotically optimal [13]. Xin [44] proposed the capped order-up-to policy, which is asymptotically optimal for lost sales inventory models. We refer to the capped order-up-to policy as (s, S, r), where r is the order cap.

More recent work on serial supply chains includes [14] and [26]. Huh et al. [14] study a periodically reviewed multi-echelon serial inventory system with capacity constraints on the order quantity at each echelon.

Multi-echelon supply chain models can have a tree like structure, with the possibility that several installations have the same supplier, but with the restriction that no installation has two different suppliers [6]. The two-echelon, one-warehouse multiple-retailers model (OWMR), where multiple local warehouses are supplied by one central warehouse, is of this kind. In [17] this model is studied with full backlogging and without emergency shipments, lateral shipments, and capacity constraints. In [9] this model is studied under other conditions, namely with backlogging, emergency shipments, lateral shipments, and without capacity constraints, but taking into account service-level agreements. Kutanoglu [20] studies such a model taking into account time-based service-level agreements that the warehouses collectively have to satisfy.

Cavalieri [5] lists four standard replenishment policies for spare parts inventory control:

- (s, Q), with fixed reorder point (s) and fixed order quantity (Q) under continuous review.
- (s, S), with fixed re-order point (s) and order-up-to level (S) under continuous or periodic review.
- (T, S), with fixed ordering interval (T) and order-up-to level (S) under periodic review.
- $(S - 1, S)$, with order-up-to level (S) in a one-for-one replenishment under continuous review, also known as the base-stock policy.

2.5 Reinforcement Learning and Spare Part Networks

The applications of reinforcement learning to spare parts networks are typically very problem specific. Therefore we cannot give a general indication. Instead we will provide references to a few interesting applications.

One of the early applications of reinforcement learning in supply chain networks is done by [39]. They applied approximate policy iteration to a one warehouse multiple retailers inventory model. In two case studies they show that RL outperforms the order-up-to heuristic.

In [26] Q-learning using neural networks for function approximation is applied to a decentralized multi-agent serial supply chain. The proposed algorithm is a modified version of the DQN algorithm in [23].

In [18], the authors investigate the performance of REINFORCE in supply chain optimization. They use a (s, Q) policy as a baseline for performance evaluation. In a simple situation, consisting of one warehouse, the REINFORCE agents outperform the baseline policy. In a more complex situation, consisting of three warehouses, a modified version of REINFORCE outperforms the baseline policy, indicating that the REINFORCE agent is 'a viable option to tackle the supply chain optimization environment' [18].

In [10] value based an policy based learning are combined. The method is based on an actor who develops a policy that is evaluated by a critic. This is more natural to RL as here there is a penalty associated with wrong choices. RL is applied to three inventory problems: lost sales, dual sourcing and multi-echelon models.

The challenge in this thesis is to translate one or several RL frameworks to fit our application.

Chapter 3

A Stylized Model

As a starting point, consider a simplified single-item spare parts inventory model with one regional distribution center (RDC) with infinite stock and one local distribution center (LDC) replenished by the RDC, see Figure 3.1. Customer demand only occurs at the LDC and is immediately met if possible. If the LDC does not have enough spare parts available, there will be an emergency shipment from the RDC at additional cost. Backlogging of unfulfilled orders is therefore not needed.

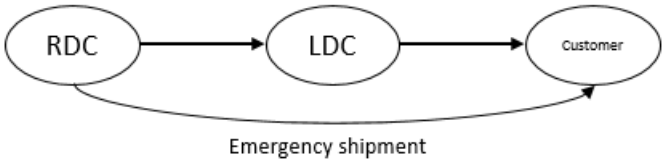


Figure 3.1: Stylized model flow chart.

The stylized model is a serial supply chain consisting of two echelons. However, since the RDC is assumed to have infinite stock, this model can also be formulated as a single-echelon model with an infinite supplier, like in [17]. Because of the use of emergency shipments this model is similar to a lost sales model as discussed in [10, 46], where a penalty is paid for all lost sales. We assume that the lead time from RDC to LDC is fixed. We consider the situation at the LDC, since that is where the decisions for replenishment need to me made.

The stylized model can mathematically be formulated as an MDP and it can be solved to optimality using dynamic programming. The MDP formulation and the optimality criterion will be given in Section 3.1. In Section 3.2 we will briefly discuss two dynamic programming approaches. These two approaches will later be connected to the RL approaches in Chapters 4 and 5. The optimal MDP solution will be used as a baseline. In Section 3.4 we will provide an example with the optimal policy and corresponding values per state obtained via dynamic programming. This example will be used in Chapters 4 and 5 as well to compare the RL results with the optimal result as presented in Section 3.4.

3.1 Finite Markov Decision Process

The stylized model is very similar to the single-product stochastic inventory control model discussed by Puterman in [27]. Here the MDP model is formulated following the steps and notation in [27] very closely. The assumptions for the stylized model are the following.

1. The decision to order spare parts at the RDC is made at the beginning of each time period. Ordered spare parts are delivered at the start of the next time period, i.e. the lead time from RDC to LDC equals one time period.
2. The customer demand distribution is i.i.d. over time.
3. If the demand exceeds the inventory level, and therefore the demand cannot be met by the LDC, there will be an emergency shipment from the RDC at additional cost.
4. The costs only depend on the current state and action, and not on time.
5. Products are sold in whole units.
6. The warehouse has a maximum capacity of s_{max} units, and a maximum of $a_{max} \leq s_{max}$ units can be ordered at each decision epoch.

Let S_t be an integer number representing the inventory level at time t at the LDC. The number of spare parts ordered at time t at the RDC, denoted by a_t, arrive just before time $t+1$. The random customer demand, D_t, occurs right after the decision on a_t is made. In the stylized model the inventory level at time $t+1$, S_{t+1}, can now be defined as

$$S_{t+1} = \min\{\max\{0, S_t - D_t\} + a_t, s_{max}\}. \tag{3.1}$$

Note that the customer demand can only be met using the inventory level at time t since the newly ordered spare parts have not yet arrived when the demand occurs, see Figure 3.2 for a schematic representation of the events within a time period. The inventory cannot become negative, since the excess demand is met using an emergency shipment. Furthermore the inventory level cannot exceed the maximum capacity s_{max}.

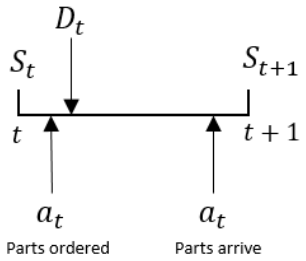


Figure 3.2: Events in each time period in the stylized model.

Next, the cost structure is defined. The costs are comprised of holding cost, ordering cost and emergency cost. These costs will be further specified one by one. The holding costs for holding x items in stock at the LDC is represented by the increasing function $H(x)$. Here it is assumed that the holding cost is linear in the number of items in stock. Hence, $H(x) = h \cdot x$, where $h > 0$ is a constant holding cost per item per time unit. The cost of ordering x spare parts at the RDC

is denoted by $O(x)$. It consists of a fixed cost $\tau_f \geq 0$ for transportation of the spare parts and a variable cost $c(x) = \tau_v \cdot x$ that increases with the ordered quantity x, where $\tau_v > 0$ is the cost per item. Hence,

$$O(x) = \begin{cases} 0 & \text{if } x = 0, \\ \tau_v \cdot x + \tau_f & \text{if } x \geq 1. \end{cases} \tag{3.2}$$

The emergency cost for excess demand x is represented by the increasing function $E(x)$, which is assumed to be linear in the excess demand, i.e. $E(x) = e \cdot x$, where $e \geq 0$ is the emergency cost per item. We make the natural assumption that the emergency cost per item is greater than the transportation cost per item, because otherwise the use of the LDC would never pay off. In case the fixed transportation cost $\tau_f = 0$ this would imply $p > \tau_v$, but for $\tau_f > 0$ this inequality is not straightforward, due to the variability of the average cost per item. To overcome this problem we set $p > \tau_f + \tau_v$.

We do not consider cost to deliver spare parts to the customer, as we do not want those cost to affect the use of the LDC. We could adjust the emergency cost to implicitly take into account costs for delivery to the customer.

The one-step reward depends on the state S_t, action a_t and demand D_t

$$R(S_t, a_t, D_t) = -H\left((S_t - D_t)^+\right) - E\left((D_t - S_t)^+\right) - O(a_t). \tag{3.3}$$

To ease the notation we sometimes write R_t instead of $R(S_t, a_t, D_t)$. The next state S_{t+1} follows automatically from Equation (3.1) when S_t, a_t, and D_t are known.

For future calculations it is more convenient to work with the expected rewards, denoted by $r_t(S_t, a_t) = \mathbb{E}[R(S_t, a_t, D_t) \mid S_t, a_t]$. The expected value is derived by conditioning on the value of D_t

$$\begin{aligned} r_t(S_t, a_t) &= -\mathbb{E}[H((S_t - D_t)^+) + E((D_t - S_t)^+) \mid S_t, a_t] - O(a_t) && (3.4) \\ &= h \cdot \mathbb{E}[D_t] + e \cdot S_t - (e+h) \cdot S_t \cdot \mathbb{P}(D_t \leq S_t) && (3.5) \\ &\qquad - (e+h) \sum_{k=S_t+1}^{\infty} k \cdot \mathbb{P}(D_t = k) - O(a_t) \\ &= -e \cdot \mathbb{E}[D_t] + e \cdot S_t - (e+h) \cdot S_t \cdot \mathbb{P}(D_t \leq S_t) && (3.6) \\ &\qquad + (e+h) \sum_{k=0}^{S_t} k \cdot \mathbb{P}(D_t = k) - O(a_t). \end{aligned}$$

The MDP can now be defined as follows.

Decision epochs:

$$\mathcal{T} = \{1, 2, \dots\}. \tag{3.7}$$

State space: the inventory level at the beginning of each time period

$$\mathcal{S} = \{0, 1, \dots, s_{max}\}. \tag{3.8}$$

Action space: the number of spare parts ordered at the RDC at the beginning of each time period

$$\mathcal{A} = \{0, 1, \dots, a_{max}\}, \tag{3.9}$$

where we assume $a_{max} \leq s_{max}$.

Expected rewards: for $t = 1, 2, \ldots$

$$r_t(S_t, a_t) = -e \cdot \mathbb{E}[D_t] + e \cdot S_t - (e + h) \cdot S_t \cdot \mathbb{P}(D_t \leq S_t) \tag{3.10}$$

$$+ (e + h) \sum_{k=0}^{S_t} k \cdot \mathbb{P}(D_t = k) - O(a_t). \tag{3.11}$$

Transition probabilities: the probability to go from state $S_t = s \in \mathcal{S}$ to state $S_{t+1} = j \in \mathcal{S}$ when taking action $a_t = a \in \mathcal{A}$ is defined as

$$p^a(s, j) = \mathbb{P}(S_{t+1} = j \mid S_t = s, a_t = a) = \mathbb{1}\{j > a\}\mathbb{P}(D_t = s - j + a) + \mathbb{1}\{j = a\}\mathbb{P}(D_t \geq s) + \mathbb{1}\{j = s_{max}\}\mathbb{P}(D_t \leq s - j + a - 1). \tag{3.12}$$

By definition $\mathbb{P}(D_t = k) = 0$ for $k < 0$. Since the demand distribution and the taken action are assumed to be stationary the transition probabilities are stationary as well.

3.1.1 The Optimality Criterion

The actions of the decision maker are completely determined by a policy π. A stationary policy is defined as the distribution over actions within each state

$$\pi(a \mid s) = \mathbb{P}(a_t = a \mid S_t = s). \tag{3.13}$$

Under suitable assumptions on the states, actions, rewards and transition probabilities, which will be specified later in this section, there exists an optimal policy in the class of stationary policies.

The goal of the decision maker is to maximize the expected total discounted reward. Let G_t^π represent the discounted sum of rewards under policy π from time-step $t \geq 0$ onward.

$$G_t^\pi = \sum_{k=0}^{\infty} \gamma^k R(S_{t+k}, a_{t+k}, D_{t+k}), \tag{3.14}$$

where $R(S_{t+k}, a_{t+k}, D_{t+k})$ is defined in (3.3) and $\gamma \in [0, 1)$ is the discount factor representing the present value of future rewards. The closer γ is to 1, the more far-sighted the evaluation is.

Let $V_t^\pi(s)$ represent the expected total discounted reward under policy π from time $t \geq 0$ onward when $S_t = s$. This is also referred to as the state value of state $s \in \mathcal{S}$ at time t, when acting according to policy π and is defined as

$$V_t^\pi(s) = \mathbb{E}^\pi[G_t^\pi \mid S_t = s] \tag{3.15}$$

$$= \mathbb{E}^\pi[\sum_{k=0}^{\infty} \gamma^k R(S_{t+k}, a_{t+k}, D_{t+k}) \mid S_t = s] \tag{3.16}$$

$$= \mathbb{E}^\pi[R(S_t, a_t, D_t) + \sum_{k=1}^{\infty} \gamma^k R(S_{t+k}, a_{t+k}t, D_{t+k}) \mid S_t = s] \tag{3.17}$$

$$= \mathbb{E}^\pi[R(S_t, a_t, D_t) + \gamma \sum_{k=0}^{\infty} \gamma^k R(S_{t+k+1}, a_{t+k+1}, D_{t+k+1}) \mid S_t = s] \tag{3.18}$$

$$= \mathbb{E}^\pi[R(S_t, a_t, D_t) + \gamma G_{t+1}^\pi \mid S_t = s]. \tag{3.19}$$

Conditioning on $a_t = a$ using (3.13), then conditioning on $S_{t+1} = j$ using (3.12), and using the Markov property results in

$$V_t^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s)\mathbb{E}^\pi[R(S_t, a_t, D_t) + \gamma G_{t+1}^\pi \mid S_t = s, a_t = a] \tag{3.20}$$

$$= \sum_{a\in\mathcal{A}} \pi(a \mid s)\Big(\mathbb{E}^\pi[R(S_t, a_t, D_t) \mid S_t = s, a_t = a] + \gamma\mathbb{E}^\pi[G_{t+1}^\pi \mid S_t = s, a_t = a]\Big) \tag{3.21}$$

$$= \sum_{a\in\mathcal{A}} \pi(a \mid s)\Big(r_t(s, a) + \gamma \sum_{j\in\mathcal{S}} p^a(s, j)\mathbb{E}^\pi[G_{t+1}^\pi \mid S_{t+1} = j]\Big) \tag{3.22}$$

$$= \sum_{a\in\mathcal{A}} \pi(a \mid s)\Big(r_t(s, a) + \gamma \sum_{j\in\mathcal{S}} p^a(s, j)V_{t+1}^\pi(j)\Big). \tag{3.23}$$

Equation (3.23) is known as the Bellman expectation equation, where $r_t(s, a)$ is defined in Equation (3.10).

In the infinite-horizon setting we are interested in

$$V^\pi(s) = \lim_{t\to\infty} V_t^\pi(s), \tag{3.24}$$

assuming that this limit exists. Puterman states that under the following assumptions the limit in Equation (3.24) can be passed to in Equation (3.23) [27, p. 146].

Assumption 3.1.1. *Stationary expected rewards, $r_t(s, a) = r(s, a)$ for all $t \geq 0$, and stationary transition probabilities, $p^a(s, j)$, that satisfy the Markov property.*

Assumption 3.1.2. *Bounded expected rewards, $|r(s, a)| \leq M < \infty$ for all $a \in \mathcal{A}$ and $s \in \mathcal{S}$.*

Assumption 3.1.3. *Discounting according to discount factor $\gamma \in [0, 1)$.*

Assumption 3.1.4. *Discrete state space $\mathcal{S}$ and discrete action space $\mathcal{A}$ are both finite.*

All assumptions hold for the stylized model[1], therefore we obtain the stationary Bellman expectation equation

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s)\Big(r(s, a) + \gamma \sum_{j\in\mathcal{S}} p^a(s, j)V^\pi(j)\Big). \tag{3.25}$$

The optimal policy is the policy specifying the best performance in the MDP, which results in optimal value function

$$V^*(s) = \sup_\pi V^\pi(s). \tag{3.26}$$

If the supremum is attained then there exists a deterministic stationary policy that is optimal [27, Thm. 6.2.7].

The Bellman optimality equation is then given by

$$V^*(s) = \sup_{a\in\mathcal{A}} \Big\{r(s, a) + \gamma \sum_{j\in\mathcal{S}} p^a(s, j)V^*(j)\Big\} \tag{3.27}$$

[1] See Appendix A for the details on Assumption 3.1.2

3.1.2 Discount and Average Optimal

In [30] it is shown that under certain conditions there exists a stationary policy which is average optimal and limit discount optimal. We will state the theorem as a translated version to the notation of the stylized model and show that the relevant conditions are satisfied.

Condition (A) The set of possible actions is finite for each state $s \in \mathcal{S}$.

Condition (B)

$$\sup_{\gamma<1}\left\{\sup_{x\in\mathcal{S}} V^*(x) - V^*(s)\right\} < \infty \quad \text{for } s \in \mathcal{S}. \tag{3.28}$$

Note that V^* implicitly depends on γ.

Theorem 3.1.1. *Assume (A) and (B). Then there exists a stationary policy which is average optimal and limit discount optimal. Furthermore*

$$g = \lim_{\gamma\to 1}(1-\gamma)\sup_{x\in\mathcal{S}} V^*(x) = \lim_{\gamma\to 1}(1-\gamma)V^*, \tag{3.29}$$

where g is assumed to be the optimal average expected reward per time step.

Instead of condition (A) the original theorem in [30] states another condition (S) that is satisfied for finite action space. Therefore we state the simplified condition (A) here. For finite action space condition (S) in [30] is satisfied. One can use the Bellman optimality Equation (3.27) to show that condition (B) is satisfied.

From Theorem 3.1.1 it now follows that there exists a policy that is both average optimal and limit discount optimal. Equation (3.29) implies that for $\gamma \to 1$, $(1-\gamma)V^*$ approximately equals the optimal average expected reward per time-step.

The average expected reward per time-step is defined as

$$\lim_{T\to\infty}\frac{1}{T}\mathbb{E}^{\pi}[\sum_{t=1}^{T} R(S_t, a_t, D_t)]. \tag{3.30}$$

3.2 Dynamic Programming

The stylized model can be solved to optimality by means of dynamic programming. Two dynamic programming algorithms were implemented: policy iteration and value iteration. The implementation of both algorithms is done based on the algorithms in [32], the pseudo code is provided in Appendix C. The resulting optimal policy and value function will be used as a baseline. The results from the RL approaches, which will be discussed in Chapters 4 and 5, will be compared to this baseline, in order to quantify the performance.

3.2.1 Policy Iteration

In the policy iteration algorithm a given policy is first evaluated after which the policy is improved. This process continues until the policy no longer changes in the improvement step. The threshold replenishment policies from the literature are all deterministic policies, so we will solely consider deterministic policies here. A finite MDP has a finite number of deterministic policies, therefore this process converges to an optimal policy within the class of deterministic policies and the optimal value function in a finite number of iterations [32].

Policy evaluation is done using the Bellman expectation equation, defined in Equation (3.23), to approximate the values $V^{\pi}(s)\,\forall s \in \mathcal{S}$. Note that the policies in policy evaluation are deterministic, therefore the Bellman equations simplify to

$$V(s) = r(s, \pi(s)) + \gamma \sum_{j \in \mathcal{S}} p^{\pi(s)}(s, j) V(j). \tag{3.31}$$

The approximation of the value function is then used for the policy improvement, which is done by calculating

$$\arg\max_{a \in \mathcal{A}} V^{a}(s) \tag{3.32}$$

for each state. The policy for state $s \in \mathcal{S}$ will be updated by setting the action resulting in the maximum value for state s. If multiple actions result in the maximum value, the smallest action, i.e. the smallest order quantity, is taken.

3.2.2 Value Iteration

The value iteration algorithm uses the Bellman optimality equation, as defined in Equation (3.27), to approximate the values $V^{\pi}(s)\ \forall s \in \mathcal{S}$. The values are iteratively updated until the value change of two consecutive sweeps is smaller than a predetermined threshold $\theta > 0$.

The value iteration algorithm takes the maximum value over all actions per state. If multiple actions result in the maximum value the smallest action, i.e. the smallest order quantity is chosen.

3.3 Simulation Environment

A simulation environment is used to mimic the behavior of the environment and measure the performance of policies. The basic simulation, presented in Algorithm 1, is based on a model environment and a policy. The simulation will also be used to sample trajectories to learn from when applying REINFORCE in Chapter 4 or Q-learning in Chapter 5.

Algorithm 1 Basic Simulation

```
1: function SIMULATION(ModelEnv, Policy)
2:     for Each trajectory do
3:         Set start state
4:         for Each time-step in the trajectory do
5:             Select action according to the policy for the current state
6:             Take action
7:             Observe reward and next state from ModelEnv
8:         Calculate and save discounted sum of rewards
```

Algorithm 2 shows the model environment for the stylized model. First the state space, action space, demand distribution and the cost parameters are initialized. The STEP function is used to take an action and returns the direct reward and the next state. The RESET function is used to set the start state within the environment.

Algorithm 2 Class StylizedModelEnv

function INIT($s_{max}, a_{max}, D, h, e, \tau_f, \tau_v$)
    Initialize the state space, action space, demand distribution and cost parameters.
function STEP(action A_t)
    Given current state S_t, action A_t and demand D_t, determine next state S_{t+1} according to Eq. (3.1).
    Determine immediate reward R_t according to Eq. (3.3).
    return R_t, S_{t+1}
function RESET
    Set start state $S_0 = 0$.
    return S_0

3.4 Example for Performance

To illustrate the performance of the investigated algorithms, one example will be used throughout this report. The parameter settings are presented in Table 3.1.

$\mathcal{S} = \{0, 1, 2, 3\}$	$h = 1$
$\mathcal{A} = \{0, 1, 2, 3\}$	$e = 10$
$D \sim \text{Pois}(1)$	$\tau_v = 1$
$\gamma = 0.99$	$\tau_f = 2$

Table 3.1: Parameter settings as an example to illustrate performance of the algorithms throughout this report.

For this example the optimal policy, obtained via dynamic programming, is

$$\pi^* = [3, 3, 2, 0]. \tag{3.33}$$

When the inventory level equals 0 or 1 the decision maker orders 3 parts, when the inventory level equals 2 the decision maker orders 2 parts, and when the inventory level equals 3 the decision maker does nothing. This is a capped order-up-to policy, with re-order point 2, order-up-to level 4 and cap 3. It might be counter intuitive that the optimal policy is to order up to inventory level 4, as the capacity is limited by 3. However, since the demand rate is 1, on average one extra part will be sold after the moment of decision. If the demand happens to be zero the over-ordered parts will be lost.

The values per state corresponding to this policy are

$$V^* = [-473.380, -467.426, -465.054, -463.010]. \tag{3.34}$$

We will use this example as a baseline for performance analysis of the RL approaches in Chapters 4 and 5.

Chapter 4

REINFORCE

The first RL method we have investigated is REINFORCE. The REINFORCE algorithm [43] directly produces a policy. The result is therefore easy to interpret. In terms of dynamic programming one can compare REINFORCE with a policy iteration approach, as that approach also directly produces a policy.

REINFORCE is a policy gradient method, in which the policy will be represented in terms of a feedforward neural network. A neural network can be interpreted as a function that maps a set of inputs to a set of outputs, it maps the set of states (or one state) to a set of actions, thus representing a policy. The policy is a parameterization in terms of the weights of the neural network, we will refer to this neural network as the policy estimator. The resulting estimated policy is a probability distribution over the actions per state and is therefore stochastic. Although the estimated policy can approach a deterministic policy, it is not allowed to become deterministic to ensure exploration [32]. The value function can, but does not need to, be used to learn the policy. Either way, once the policy is learned the value function is not needed for action selection.

In each learning step the weights of the policy estimator are updated via gradient descent for a loss function, which will be defined later in this chapter. To calculate the loss, the discounted sum of rewards needs to be calculated. To calculate the discounted sum of rewards from a certain time-step onward, an entire trajectory has to be simulated. REINFORCE is therefore a Monte Carlo method. To simulate the trajectories, a simulation environment was set up as explained in Section 3.3. The action selection is done according to the policy resulting from the policy estimator. As REINFORCE follows its own learned policy, it is an on-policy method.

Figure 4.1 shows how the different elements in the REINFORCE method are connected. The simulation environment is used to sample trajectories of a pre-specified length. Within each trajectory, actions are selected according to a policy resulting from the policy estimator based on the most recent values for the weights. For each trajectory the discounted sum of rewards will be calculated. Then the loss function can be calculated, based on which the parameters of the policy estimator will be updated via gradient descent. This process repeats for a pre-specified number of iterations.

In the remainder of this chapter we will discuss the algorithm and its main elements in Section 4.1. Then we elaborate on the challenges that we encountered in Section 4.2. This chapter will be concluded with an example followed by numerical results in Section 4.3.

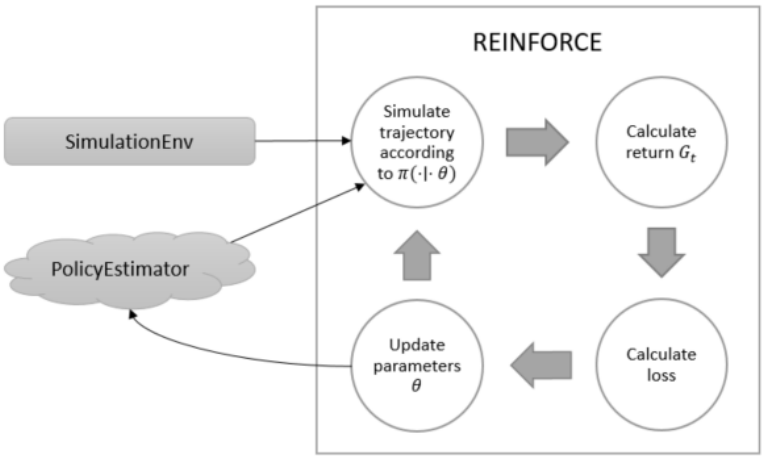


Figure 4.1: Overview structure REINFORCE.

4.1 Mathematical Formulation

The REINFORCE algorithm is specified in Algorithm 3. In this section we provide a brief explanation of the main elements.

Algorithm 3 REINFORCE

Input: The model environment and the neural network for estimating the policy distribution.
Output: A list containing the total reward for each trajectory and the final policy according to PolicyEstimator.

1: **function** REINFORCE(environment, PolicyEstimator)
2: **for** a total of N trajectories **do**
3: Generate trajectory $S_0, A_0, R_0, \ldots, S_{T-1}, A_{T-1}, R_{T-1}$ according to policy $\pi(\dot{|};\theta)$
4: Calculate discounted sum of rewards $G^{\pi}_{t,T}$ for $t = 0, 1, \ldots, T-1$ according to Eq. (4.1).
5: Calculate loss according to Eq. (4.2)
6: Determine gradients according to Eq. (4.3)
7: Update the parameters of the policy estimator via gradient descent
8: **return** total reward per trajectory and final policy according to the policy estimator.

Each iteration starts by simulating a trajectory of length T. This is done using the simulation environment as presented in Section 3.3, where the action selection is done according to the most recent policy resulting from the policy estimator. As was mentioned in the introduction of this chapter, the policy is a parameterization in terms of the weights of the neural network. We denote these weights by $\boldsymbol{\theta}$. For our experiments, we use a neural network with one hidden layer, consisting of four hidden nodes, and ReLU activation as policy estimator, but one can choose other network structures or activation functions. Neural networks are often used as a black box. To gain a better understanding of what happens inside this black box and of the parameterization in terms of the weights, we provide an example to illustrate what happens in the forward propagation through the neural network in Section 4.2.1.

Once the trajectory is simulated, for $t = 0, 1, \ldots, T-1$ the discounted sum of rewards is

calculated as follows

$$G_{t,T}^{\pi} = \sum_{k=0}^{T-t} \gamma^{t+k} R_{t+k}. \tag{4.1}$$

Note that this is a biased estimator for the discounted sum of rewards as defined in Equation (3.14), since we stop the trajectory after time T, instead of considering an infinite horizon. We will elaborate on this estimator in Section 4.2.2.

Next the loss is calculated. The (episodic) REINFORCE loss function [43] is defined as

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{t=0}^{T-1} G_{t,T}^{\pi} \cdot \log(\pi_{\boldsymbol{\theta}}(a_t \mid S_t)). \tag{4.2}$$

This loss function is known as the cross entropy loss. If an action that is taken with a low probability leads to a high reward, or an action that is taken with a high probability leads to a low reward, the loss will increase, indicating that the policy should be adjusted.

The corresponding gradient with respect to the weights $\boldsymbol{\theta}$ of the policy estimator is

$$\nabla\mathcal{L}(\boldsymbol{\theta}) = -\sum_{t=0}^{T-1} G_{t,T}^{\pi} \cdot \frac{\nabla\pi_{\boldsymbol{\theta}}(a_t \mid S_t)}{\pi_{\boldsymbol{\theta}}(a_t \mid S_t)}. \tag{4.3}$$

The gradient indicates the direction in which the weights should be updated, to reduce the loss function.

The intuition behind this update rule is that actions leading to high rewards are taken more often, and actions leading to low rewards are taken less often. As one can see in Equation (4.3) the update is proportional to the gradient of the probability of taking action a_t in state S_t, indicating the direction that most increases the probability of taking action a_t on future visits to state S_t for actions leading to high rewards. The inverse proportionality to the probability that a particular action is taken, prevents that frequently occurring actions (that do not necessarily lead to high rewards) dominate the results.

The challenges that we encountered regarding the loss function will be elaborated on in Section 4.2.3.

The gradient in Equation (4.3) is used to update the weights of the policy estimator via gradient descent. The basic gradient descent update is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\nabla\mathcal{L}(\boldsymbol{\theta}), \tag{4.4}$$

where $\alpha > 0$ is the step size. We will use the adaptive stochastic gradient descent method Adam. For further details on this method we refer to [19].

Note that to run this procedure one does not need to have specific knowledge of the transition probabilities and reward structure, as long as one has an oracle, such as the simulation environment, to generate trajectories for a given policy. It is also possible to use observations from an actual operating environment, but that is outside the scope of this project.

4.2 Challenges

When implementing the algorithm, we encountered several challenges. The first challenge was to gain a better understanding of the policy as a parameterization in terms of the policy estimator

weights. This was done by means of an example in Section 4.2.1. The second challenge was to gain a better understanding of the nature of the loss function in Equation (4.2). To this end we first need to understand how the policy is a parameterization in terms of the weights of the policy estimator.

Before we elaborate on the nature of the loss function in Section 4.2.3 we first elaborate on the biased estimator for the discounted sum of rewards in Section 4.2.2, were we propose a correction factor to reduce the bias.

4.2.1 Policy Parameterization Example

In this section we provide an example to gain a better understanding of the policy as a parameterization of the weights of the policy estimator.

Consider a small neural network as presented in Figure 4.2. The network takes one single value as input, the current inventory level. There is one hidden layer consisting of two hidden nodes with ReLU activation, represented by x_i and y_i respectively for $i = 1, 2$. The output layer is represented by z_i for $i = 1, 2$ and the Softmax over two possible actions is the final output of the network.

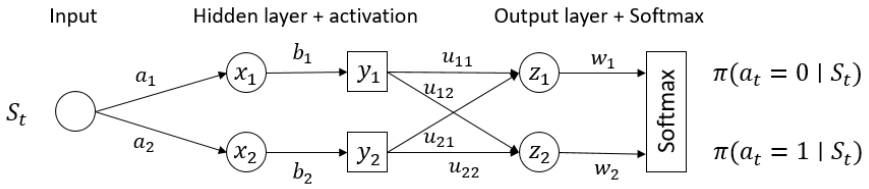


Figure 4.2: Example neural network as policy estimator

Each arrow in the schematic representation represents an assigned weight that has to be learned by the neural network. The total weight vector in this network is

$$\boldsymbol{\theta} = (a_1, a_2, b_1, b_2, u_{11}, u_{12}, u_{21}, u_{22}, w_1, w_2). \tag{4.5}$$

The action probabilities can be expressed in terms of these weights and the input state. The forward propagation throughout the network is given step by step.

First the weights a_1 and a_2 are assigned to the input value, in the hidden linear layer we have

$$x_1 = a_1 \cdot S_t, \tag{4.6}$$
$$x_2 = a_2 \cdot S_t. \tag{4.7}$$

Then the biases b_1 and b_2 are added to x_1 and x_2 respectively and after applying the ReLu function we have

$$y_1 = \max(a_1 \cdot S_t + b_1, 0), \tag{4.8}$$
$$y_2 = \max(a_2 \cdot S_t + b_2, 0). \tag{4.9}$$

Next, weights are assigned to y_1 and y_2 and in the linear output layer we find

$$z_1 = u_{11} \cdot y_1 + u_{21} \cdot y_2 = u_{11} \max(a_1 \cdot S_t + b_1, 0) + u_{21} \max(a_2 \cdot S_t + b_2, 0), \tag{4.10}$$
$$z_2 = u_{12} \cdot y_1 + u_{22} \cdot y_2 = u_{12} \max(a_1 \cdot S_t + b_1, 0) + u_{22} \max(a_2 \cdot S_t + b_2, 0). \tag{4.11}$$

As a final step the biases w_1 and w_2 are added to z_1 and z_2 respectively and the Softmax function is applied. This results in

$$\pi_{\boldsymbol{\theta}}(0 \mid S_t) = \frac{\exp(z_1 + w_1)}{\exp(z_1 + w_1) + \exp(z_2 + w_2)}, \tag{4.12}$$

$$\pi_{\boldsymbol{\theta}}(1 \mid S_t) = \frac{\exp(z_2 + w_2)}{\exp(z_1 + w_1) + \exp(z_2 + w_2)}. \tag{4.13}$$

4.2.2 Discounted Sum of Rewards

Here we study the estimator for the discounted sum of rewards as defined in Equation (4.1). Since we stop the trajectory after time T, and thus have a finite instead of an infinite time-horizon, this estimator is biased. To make sure the bias is small, one can determine the total trajectory length T that needs to be simulated in order to lose a small amount (at most $\varepsilon > 0$) in the approximation of $V^\pi(s)$, defined in Equation (3.14). Note that the total trajectory length that is needed depends on γ.

Using

$$\mathbb{E}^\pi[G^\pi_{t,T} \mid S_t = s] = \sum_{k=0}^{T-t-1} \gamma^k \mathbb{E}^\pi[R(S_{t+k}, a_{t+k}, D_{t+k} \mid S_t = s] \tag{4.14}$$

as an estimator for $V^\pi(s)$, we lose

$$\sum_{k=T-t}^{\infty} \gamma^k \mathbb{E}^\pi[R(S_{t+k}, a_{t+1}, D_{t+k} \mid S_t = s] \tag{4.15}$$

in the tail. The expected reward $\mathbb{E}^\pi[R(S_{t+k}, a_{t+1}, D_{t+k} \mid S_t = s]$ is bounded for all $k = 0, 1, \ldots$. We have

$$|\mathbb{E}^\pi[R(S_{t+k}, a_{t+1}, D_{t+k}) \mid S_t = s]| \leq C, \tag{4.16}$$

for some $C > 0$. We refer the reader to Appendix A for the details on how to bound the expected reward.

Then

$$|\sum_{k=T-t}^{\infty} \gamma^k \mathbb{E}^\pi[R(S_{t+k}, a_{t+1}, D_{t+k} \mid S_t = s]| \leq \frac{\gamma^{T-t}}{1-\gamma} C < \varepsilon \tag{4.17}$$

results in remaining trajectory length

$$T - t > \frac{\log(\varepsilon(1-\gamma)/C)}{\log(\gamma)}. \tag{4.18}$$

Instead of sampling a long enough trajectory to make sure the bias is small, one can also reduce the bias by applying a suitable correction factor. We propose a correction factor for $\mathbb{E}^\pi[G^\pi_{t,T} \mid S_t = s]$, depending on the remaining trajectory length $T - t$.

Suppose that the reward in each time-step equals 1. The discounted sum of rewards at an arbitrary time t in the infinite-horizon setting, denoted by G^1_t, is then

$$G^1_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}. \tag{4.19}$$

If we generate a trajectory of length T, then the estimator for the discounted sum of rewards at an arbitrary time t, without correction for the remaining trajectory length, equals

$$G^1_{t,T} = \sum_{k=0}^{T-t-1} \gamma^k = \frac{1-\gamma^{T-t}}{1-\gamma}. \tag{4.20}$$

Applying a correction factor

$$\frac{1}{1-\gamma^{T-t}} \tag{4.21}$$

results in the discounted sum of rewards for the infinite time-horizon setting. Although the one-step reward to be a random variable instead of being equal to 1 in each time-step, this correction factor is still useful. In fact, if the process is in stationarity at time t, such that $\mathbb{E}^\pi[R(S_{t+k}, a_{t+k}, D_{t+k}]$ equals the stationary expected reward $\mathbb{E}^\pi[R(S, a, D)]$, then multiplying $\mathbb{E}^\pi[G^\pi_{t,T}]$ with the correction factor approximately yields the expected infinite-horizon discounted rewards for $\gamma \to 1$, i.e. provides an unbiased estimator.

We relate this to the stylized model using simulation results for the example in Section 3.4, were we compare the uncorrected discounted rewards with the corrected discounted rewards and with the value per state, which equals the expected discounted sum of rewards given the start state in the infinite-horizon setting.

The results for a total trajectory length T equal to 100, 500, and 1000 are presented in Table 4.1. To obtain these results 10.000 episodes were simulated per trajectory length. For each episode we keep track of the uncorrected discounted sum of rewards and the corrected discounted sum of rewards at $t = 0$, given the starting state. For each state the (un)corrected discounted sum of rewards were averaged.

For a long remaining trajectory length the time correction has little impact, as one can see from the results for remaining trajectory length 1000. The uncorrected discounted sum of rewards and the corrected discounted sum of rewards are very close (the confidence intervals almost entirely overlap). For a remaining trajectory of length 100 one can see that the corrected discounted sum of rewards is closer to the values than the uncorrected discounted sum of rewards. However, the correction is still a bit off, as the values resulting from dynamic programming do not lie within the 95% confidence intervals of the corrected discounted sum of rewards per state.

Another possibility to avoid the cut-off effect in a finite-horizon setting compared to an infinite-horizon setting is to sample a longer trajectory than will actually be used. This is relatively easy to implement: sample a longer trajectory, calculate the discounted rewards based on the sampled trajectory, delete the last x time steps of the trajectory and use that in the rest of the algorithm. The 'extra' x time steps one needs to sample can be determined by a similar analysis as is done in Table 4.1. A good choice for these parameter settings is 500, as the value per state resulting from dynamic programming lies within the confidence interval of the corrected discounted sum of rewards. Obviously 100 extra time steps is not enough, as there is still bias in the corrected discounted sum of rewards. 1000 would definitely be enough, but sampling a longer trajectory takes more time. Therefore one does not want to choose such a large number of extra steps.

Note that this example is for the parameter settings in Section 3.4 specifically. For other parameter settings, one needs another trajectory length to reduce the bias caused by the cut-off of trajectories when using the correction factor.

Due to randomness in the demand, and therefore in the rewards, the variance of the discounted sum of rewards can be quite high. A lower variance in the discounted returns results in more

(a) Results for total trajectory length equal to 1000

State	V^π	$\overline{G}_0$ (CI)	$\overline{G}_0^{corr}$ (CI)
0	-473.38	-473.19 (-474.47 , -471.91)	-473.21 (-474.49 , -471.93)
1	-467.42	-468.52 (-469.80 , -467.24)	-468.54 (-469.82 , -467.26)
2	-465.05	-464.87 (-466.09 , -463.64)	-464.89 (-466.11 , -463.66)
3	-463.01	-463.35 (-464.59 , -462.11)	-463.37 (-464.61 , -462.13)

(b) Results for total trajectory length equal to 500

State	V^π	$\overline{G}_0$ (CI)	$\overline{G}_0^{corr}$ (CI)
0	-473.38	-470.59 (-471.87 , -469.31)	-473.71 (-474.99 , -472.42)
1	-467.42	-464.24 (-465.49 , -462.99)	-467.31 (-468.57 , -466.06)
2	-465.05	-461.84 (-463.09 , -460.59)	-464.89 (-466.15 , -463.63)
3	-463.01	-459.98 (-461.19 , -458.77)	-463.02 (-464.24 , -461.81)

(c) Results for total trajectory length equal to 100

State	V^π	$\overline{G}_0$ (CI)	$\overline{G}_0^{corr}$ (CI)
0	-473.38	-303.33 (-304.50 , -302.16)	-478.46 (-480.31 , -476.62)
1	-467.42	-297.90 (-299.10 , -296.71)	-469.90 (-471.78 , -468.02)
2	-465.05	-295.01 (-296.17 , -293.86)	-465.34 (-467.17 , -463.52)
3	-463.01	-293.37 (-294.52 , -292.22)	-462.75 (-464.56 , -460.94)

Table 4.1: Simulation results to compare the average uncorrected and corrected discounted rewards at time $t = 0$ to the values per state, for $\gamma = 0.99$.

stabilized learning. One way to reduce the variance is by normalizing the discounted rewards by subtracting the mean and dividing by the standard deviation, this is known as whitening. For the normalization there is a case distinction for $\text{std}(G) > 0$, as the normalization would not be possible if one divides by the standard deviation when it equals 0.

4.2.3 Non-convex Loss Function

As mentioned in Section 4.1 the weights are updated via a gradient descent method. Gradient descent methods are known to converge to the optimal solution for convex functions [3], but for non-convex functions this is not guaranteed. Here we provide an example that shows that the loss function, defined in Equation (4.2), is non-convex. Due to the non-convex nature of the loss function, REINFORCE is not guaranteed to converge to the optimal solution. Nevertheless, from the literature it is known that REINFORCE has good convergence properties, but that it tends to converge to a local optimum [32].

Recall the loss function as previously defined in Equation (4.2),

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{t=0}^{T-1} G_{t,T}^{\pi} \cdot \log(\pi_{\boldsymbol{\theta}}(a_t \mid S_t)). \tag{4.22}$$

Due to the use of a neural network as policy estimator, the log-probabilities in this equation are neither convex nor concave. We show this by a counterexample and use it to show that $\mathcal{L}(\boldsymbol{\theta})$ is neither convex nor concave.

We use the example neural network from Section 4.2.1, Figure 4.2.

We define

$$L_0(S_t) = \log(\pi_{\boldsymbol{\theta}}(0 \mid S_t)), \tag{4.23}$$
$$L_1(S_t) = \log(\pi_{\boldsymbol{\theta}}(1 \mid S_t)), \tag{4.24}$$

where $\pi_{\boldsymbol{\theta}}(a \mid S_t)$ for $a = 0, 1$ is defined in Equation (4.12) and (4.13).

Then we can write the loss function as

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{t=0}^{T-1} G_{t,T}^{\pi} L_{a_t}(S_t), \tag{4.25}$$

where only $L_{a_t}(S_t)$ depends on $\boldsymbol{\theta}$.

To prove convexity (concavity) one needs the following two theorems.

Theorem 4.2.1. *[28] A twice differentiable function of several variables is convex on a convex set if and only if the Hessian matrix of second order partial derivatives is positive semi-definite on the interior of the convex set.*

Furthermore, we know that a function f is concave if and only if $-f$ is convex.

Theorem 4.2.2. *[36, Appendix C] A $n \times n$-matrix M is positive semi-definite if and only if*

$$x^T M x \geq 0 \quad \forall x \in \mathbb{R}^n$$

We can combine these two theorems in the following theorem.

Theorem 4.2.3. *A function of several variables is convex (concave) on a convex set if and only if the Hessian matrix of second order partial derivatives, H, satisfies*

$$x^T H x \geq (\leq)\ 0 \quad \forall x \in \mathbb{R}^n \tag{4.26}$$

The gradient of the loss function with respect to $\boldsymbol{\theta}$ is

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = -\sum_{t=0}^{T-1} G_{t,T}^{\pi} \nabla L_{a_t}(S_t), \tag{4.27}$$

and the Hessian with respect to $\boldsymbol{\theta}$ is

$$\nabla^2 \mathcal{L}(\boldsymbol{\theta}) = -\sum_{t=0}^{T-1} G_{t,T}^{\pi} \nabla^2 L_{a_t}(S_t). \tag{4.28}$$

If $\nabla^2 L_{a_t}(S_t)$ is neither positive nor negative semi-definite, then $\nabla^2 \mathcal{L}(\boldsymbol{\theta})$ is neither positive nor negative semi-definite, since $G_{t,T}^{\pi} \leq 0$ for all $t \geq 0$. It follows that $\mathcal{L}(\boldsymbol{\theta})$ is neither convex nor concave.

We will show that for given parameter $\boldsymbol{\theta}$ there exists a vector $x \in \mathbb{R}^{10}$ such that $x^T H x \leq 0$ and a vector $y \in \mathbb{R}^{10}$ such that $y^T H y > 0$, and therefore the Hessian matrix of the function L_0, with respect to $\boldsymbol{\theta}$, is neither positive nor negative semi-definite, from which we can conclude that the function L_0 is neither convex nor concave. For L_1 similar steps can be taken.

We will show that L_0 is not convex nor concave, for L_1 similar steps can be taken. We have

$$L_0 = z_1 + w_1 - \log\big(\exp(z_1 + w_1) + \exp(z_2 + w_2)\big), \tag{4.29}$$

where z_1 and z_2 are defined in Equation (4.10) and Equation (4.11) respectively The Hessian matrix of L_0 is defined as follows

$$H := \nabla^2 L_0 = \begin{bmatrix} \frac{\partial^2 L_0}{\partial a_1^2} & \frac{\partial^2 L_0}{\partial a_1 \partial a_2} & \cdots & \frac{\partial^2 L_0}{\partial a_1 \partial w_1} & \frac{\partial^2 L_0}{\partial a_1 \partial w_2} \\ \frac{\partial^2 L_0}{\partial a_2 \partial a_1} & \frac{\partial^2 L_0}{\partial a_2^2} & \cdots & \frac{\partial^2 L_0}{\partial a_2 \partial w_1} & \frac{\partial^2 L_0}{\partial a_2 \partial w_2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial^2 L_0}{\partial w_1 \partial a_1} & \frac{\partial^2 L_0}{\partial w_1 \partial a_2} & \cdots & \frac{\partial^2 L_0}{\partial w_1^2} & \frac{\partial^2 L_0}{\partial w_1 \partial w_2} \\ \frac{\partial^2 L_0}{\partial w_2 \partial a_1} & \frac{\partial^2 L_0}{\partial w_2 \partial a_2} & \cdots & \frac{\partial^2 L_0}{\partial w_2 \partial w_1} & \frac{\partial^2 L_0}{\partial w_2^2} \end{bmatrix}. \tag{4.30}$$

For this example we set all weights equal to 1, i.e. $\boldsymbol{\theta} = (1, 1, 1, 1, 1, 1, 1, 1, 1)$. The Hessian matrix of L_0 with respect to $\boldsymbol{\theta}$ is given in Appendix B.

For $s \in \mathcal{S}$ and vector $x = [0, 0, 0, 0, 0, 0, 0, 0, 1, -1]$ we have

$$x^T H x = -1 < 0. \tag{4.31}$$

For $s = 0$ and vector $x = [1, 0, 0, 1, 1, 0, 1, 0, 0, 1]$ we have

$$x^T H x = \frac{3}{4} > 0. \tag{4.32}$$

From this we can already conclude that the Hessian of L_0 is neither positive nor negative semi-definite. A similar example can be given for L_1. This results in the Hessian of the loss function to be neither positive nor negative semi-definite. Therefore the loss function is neither convex nor concave.

For additional details we refer to Appendix B, where we provide results of a small study on several vectors and values of $s \in \mathcal{S}$.

4.3 Example

To illustrate the performance of REINFORCE we use the example from Section 3.4. The result is obtained by simulating 5000 trajectories, each of length 1000.

The final stochastic policy, where the rows represent the states and the columns represent the actions, is

$$\pi = \begin{bmatrix} 6.59 \cdot 10^{-6} & 2.18 \cdot 10^{-6} & 3.14 \cdot 10^{-4} & \mathbf{0.999} \\ 1.24 \cdot 10^{-2} & 1.61 \cdot 10^{-4} & 8.62 \cdot 10^{-3} & \mathbf{0.979} \\ \mathbf{0.996} & 2.60 \cdot 10^{-5} & 1.31 \cdot 10^{-3} & 2.95 \cdot 10^{-3} \\ \mathbf{0.999} & 2.89 \cdot 10^{-7} & 2.46 \cdot 10^{-6} & 1.11 \cdot 10^{-7} \end{bmatrix}, \tag{4.33}$$

approximating deterministic policy

$$\pi = [3, 3, 0, 0]. \tag{4.34}$$

The corresponding values, obtain via policy evaluation, are

$$V^\pi = [-487.11, -481.16, -479.41, -476.88]. \tag{4.35}$$

Figure 4.3 gives a graphical representation of the learning. Note that after approximately 3000 simulated trajectories (which takes approximately 16 minutes) the value for state 0 fluctuates around one value, $V(0) \approx -490$, and thus seems to have stabilized. Nevertheless, the resulting policy is not the optimal policy. It is very likely that the REINFORCE algorithm got stuck in a local optimum, due to non-convexity of the loss function that is to be optimized.

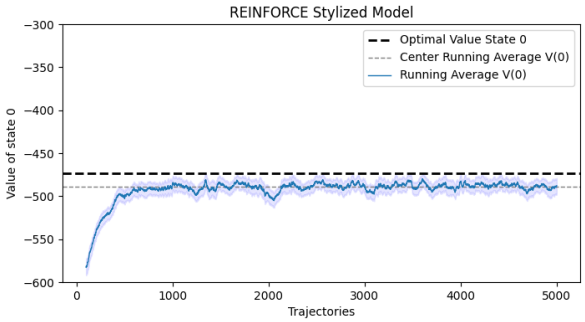


Figure 4.3: Visualization convergence REINFORCE, stylized model. The thick blue line is the running average of the value in state 0 over the 100 most recent simulated trajectories, and the lighter blue area represents the 95% confidence bound.

4.4 Concluding Remarks

In this chapter we provided more insight in the elements of REINFORCE, in particular in the loss function. We provided an example to show that the loss function is non-convex, due to which REINFORCE tends to converge to a local optimum. Furthermore we proposed a correction factor to reduce the bias in a finite trajectory estimator of the discounted sum of rewards for the infinite-horizon.

For the example in Section 3.4 REINFORCE did not find the optimal policy. Comparing the value of state 0, $V^{\pi}(0) = -487.11$, with the optimal value of state 0, $V^{*}(0) = -473.38$, the deviation is -2.9%, which is not bad. It is very likely that the method found a local optimum.

Chapter 5

Q-learning

In the previous chapter we saw that REINFORCE tends to converge to a local optimum. Therefore we investigate a second RL method. Q-learning is a value based RL method, in which the value function is updated iteratively via a specific update rule. In terms of dynamic programming one can compare Q-learning with a value iteration approach. The Q-function representing the values of state-action pairs is learned, based on which the policy can be determined. To determine the policy, one simply chooses the action with the highest Q-value for the given state. The learned policy is therefore a deterministic policy.

The simplest form of Q-learning is tabular Q-learning, where one Q-function is updated iteratively. The update rule will be given in Section 5.2. Tabular Q-learning converges in probability to the optimal policy under several conditions [41]. In stochastic MDPs, like the stylized model, Q-learning can perform poorly due to overestimation of action values. This overestimation is caused by a maximization step over the estimated action values, which tends to prefer overestimated to underestimated values [12].

To avoid this overestimation the double Q-learning algorithm was proposed in [37]. In this method, instead of one Q-function, two Q-functions are stored and iteratively updated. Both Q-functions are updated with a value from the other Q-function for the next state. We will elaborate on this method in Section 5.3.

For models with a large state and/or action space tabular Q-learning and double Q-learning require a lot of memory as the size of the Q-table representing the Q-function grows quadratically in the number of states and actions, i.e. the size of the Q-table equals $|\mathcal{S}| \cdot |\mathcal{A}|$. Also, since in each iteration only one of the Q-values is updated, it takes a long time to converge. This is where deep Q-learning might become useful. The DQN algorithm [23] combines Q-learning with a neural network to estimate the Q-function. The neural network takes the observed state as input and outputs the Q-function for each state-action pair. The DQN procedure will be discussed in Section 5.4.

5.1 Mathematical Formulation

In Q-learning the main goal is to learn the Q-function, the expected discounted sum of rewards starting in state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}$. From the Q-function the policy π can be derived. The learned Q-function directly approximates the optimal Q-function, Q^* [32], from which the

optimal policy π^* can be derived.

The outcome of the Q-function for a given state-action pair is also referred to as the Q-value or state-action value, and is denoted by $Q^\pi(s, a)$ where $s \in \mathcal{S}$ and $a \in \mathcal{A}$. The Q-function under a policy π is defined as

$$Q^\pi(s, a) = \mathbb{E}[G_t^\pi \mid S_t = s, A_t = a], \tag{5.1}$$

for all $t \geq 0$, where G_t^π is defined in Equation (3.14). The Q-function is related to the state value function V as follows,

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) Q^\pi(s, a). \tag{5.2}$$

Note that for a deterministic policy $\pi(s)$ we have

$$V^\pi(s) = Q^\pi(s, \pi(s)). \tag{5.3}$$

We want to find the policy that results in the highest Q-values for all state-action pairs,

$$Q^*(s, a) = \max_\pi Q^\pi(s, a). \tag{5.4}$$

These optimal Q-values are related to the optimal state values $V^*(s)$

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a). \tag{5.5}$$

5.2 Tabular Q-learning

In (tabular) Q-learning [40] the update rule is

$$Q_{t+1}(S_t, A_t) \longleftarrow Q_t(S_t, A_t) + \alpha_t \left[R(S_t, A_t, D_t) + \gamma \max_{a \in \mathcal{A}} Q_t(S_{t+1}, a) - Q_t(S_t, A_t) \right], \tag{5.6}$$

where $S_t \in \mathcal{S}$ is the state at time t, $A_t \in \mathcal{A}$ is the action taken at time t, $R(S_t, A_t, D_t)$ is the received reward after taking action A_t in state S_t, $\gamma \in [0, 1)$ is the discounting factor and α_t is the learning rate at time t.

The learned Q-value directly approximates the optimal Q-value, independent of the policy that is followed [32]. The Q-values are updated in every time-step.

5.2.1 Convergence Properties

Tabular Q-learning is proved to converge to the optimal policy with probability 1 [41]. The following conditions for this convergence are stated in [33].

1. Finite state and action space.
2. $\text{Var}(R(s, a, D)) < \infty$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$.
3. $\gamma < 1$
4. The learning rate α_n satisfies $\sum_{n=0}^{\infty} \alpha_n = \infty$ and $\sum_{n=0}^{\infty} \alpha_n^2 < \infty$.

All these assumptions are met by the stylized model. The first and third condition are satisfied by definition. With straightforward computations one can show that the variance of the one-step rewards is bounded, so the second condition is satisfied as well. For the fourth condition to be satisfied we have to make a clever choice for the learning rate. The learning rate can be of the form

$$\alpha_n = \frac{1}{n^\omega}, \tag{5.7}$$

for $\omega \in (\frac{1}{2}, 1]$, where n is the number of sampled trajectories. In our implementation we use $\alpha_n = \frac{1}{n^{3/5}}$. We will come back to why we made this choice in Section 5.3.2.

Besides the above mentioned conditions, there is an important implicit condition that each state-action pair is sampled infinitely often. In a finite simulation this is intrinsically impossible, but we can ensure that each state-action pair is sampled a reasonable number of times by letting ε decay linearly to a non-zero minimum value in the $\varepsilon-$greedy action selection.

Due to a minimum $\varepsilon_{min} > 0$ the agent does not act optimally during the learning process, as there is always a positive probability of taking a random action. The sub-optimality can be avoided by using Q-learning in an off-line manner, where the policy is learned before actually being applied.

5.2.2 Implementation

Algorithm 4 specifies the tabular Q-learning algorithm. The Q-function is represented by a matrix where each row represents a state and each column represents an action. In each step of a trajectory this table is updated according to Equation (5.6).

Algorithm 4 Tabular Q-learning

Input: The simulation environment
Output: The Q-function

```
1: function Q-LEARNING(environment)
2:     for each trajectory do
3:         Initialize start state S
4:         for each step in the trajectory do
5:             Choose action A under policy derived from Q (ϵ−greedy)
6:             Take action A, observe reward R and next state S′
7:             Update Q(S, A) according to (5.6)
8:             S ← S′
9:     return total reward per trajectory and final Q-function
```

To select actions we make use of an $\varepsilon-$greedy approach, which is a well known method to ensure exploration. A random action is chosen with probability ε, and with probability $1 - \varepsilon$ the current best action, i.e. the greedy action, is chosen. Usually, in reinforcement learning, the aim is to explore a lot in the beginning and explore less later in the process. Therefore ε often decays over time.

Note that one does not need specific knowledge of the transition probabilities or reward structure to run the Q-learning procedure. Like in REINFORCE the procedure uses an oracle that generates trajectories, or it is possible to use observations from an actual operating environment as input.

5.2.3 Example

To illustrate the performance of tabular Q-learning we use the example from Section 3.4. Here we present the results of a run consisting of 5000 trajectories of length 1000, resulting in the optimal policy.

The final Q-function, where the rows and columns represent the states and actions respectively, is

$$Q^{\pi} = \begin{bmatrix} -478.41 & -476.89 & -476.88 & \mathbf{-473.58} \\ -470.81 & -470.53 & -470.64 & \mathbf{-467.84} \\ -467.03 & -467.46 & \mathbf{-464.80} & -467.57 \\ \mathbf{-463.12} & -465.59 & -466.02 & -466.46 \end{bmatrix}. \tag{5.8}$$

From this Q-function we derive the optimal policy

$$\pi = [3, 3, 2, 0], \tag{5.9}$$

with corresponding values

$$\max_{a \in \mathcal{A}} Q^{\pi}(\cdot, a) = [-473.58, -467.84, -464.80, -463.12]. \tag{5.10}$$

Figure 5.1 gives a graphical representation of the learning. Note that the learned Q-value for state 0 is slightly higher than the simulated discounted sum of rewards in state 0. This is due to the ε-greedy action selection with a minimum $\varepsilon = 0.1$.

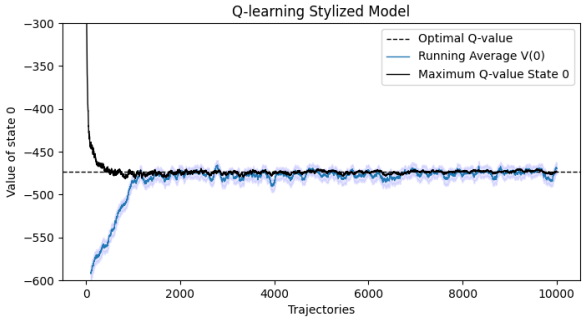


Figure 5.1: Visualization convergence Q-learning, stylized model. The black line represents the learned Q-value in state 0. The thick blue line is the running average of the simulated value in state 0 over the 100 most recent simulated trajectories, and the lighter blue area represents the 95% confidence bound.

5.3 Double Q-learning

The maximization in Equation (5.6) can lead to overestimation of the Q-values, also known as the maximization bias [32]. One way to overcome this problem of overestimation is to use double Q-learning. In the double Q-learning algorithm two Q-functions are updated simultaneously instead of updating one Q-function. To update one of the Q-functions the other Q-function is used and vice versa.

The update rules for both Q-functions are as follows,

$$Q_1(S_t, A_t) \longleftarrow Q_1(S_t, A_t) + \alpha \left[R_t + \gamma Q_2(S_{t+1}, \arg\max_{a \in \mathcal{A}} Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right], \quad (5.11)$$

$$Q_2(S_t, A_t) \longleftarrow Q_2(S_t, A_t) + \alpha \left[R_t + \gamma Q_1(S_{t+1}, \arg\max_{a \in \mathcal{A}} Q_2(S_{t+1}, a)) - Q_2(S_t, A_t) \right], \quad (5.12)$$

where $R_t = R(S_t, A_t, D_t)$.

In every time-step only one of the Q-values is updated, as governed by a fair coin-flip. The computation time will therefore remain the same as for Q-learning. The memory requirements, however, are doubled because two Q-tables are stored instead of one.

5.3.1 Implementation

The implementation is very similar to (regular) Q-learning. See Algorithm 5 for the double Q-learning algorithm.

Algorithm 5 Tabular Double $Q-$learning

Input: The simulation environment
Output: The Q-function

function Q-LEARNING(environment)
  for each trajectory **do**
    Initialize start state S
    for each step in the trajectory **do**
      Choose action A under policy derived from $Q_1 + Q_2$ ($\epsilon-$greedy)
      Take action A, observe reward R and next state S'
      Update $Q_1(S, A)$ according to (5.11) with probability 0.5
      Else update $Q_2(S, A)$ according to (5.12)
      $S \leftarrow S'$
  Final Q-function $= \frac{1}{2}(Q_1 + Q_2)$
  return total reward per trajectory and final Q-function

5.3.2 Example

To illustrate the performance of tabular double Q-learning we use the example from Section 3.4. Here we present the result of a run consisting of 15.000 trajectories of length 1000, resulting in the optimal policy.

The final Q-function, where the rows and columns represent the states and actions respectively, is

$$Q^\pi = \begin{bmatrix} -481.40 & -479.97 & -478.45 & \mathbf{-475.25} \\ -472.24 & -471.27 & -470.26 & \mathbf{-469.14} \\ -467.76 & -467.72 & \mathbf{-466.75} & -467.78 \\ \mathbf{-464.62} & -465.83 & -465.81 & -466.39 \end{bmatrix}. \quad (5.13)$$

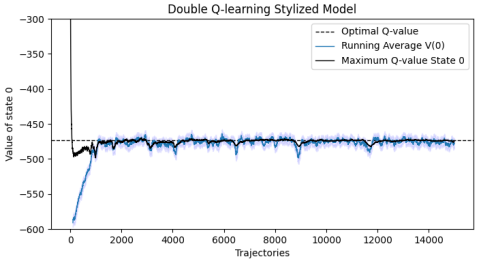
From this Q-function we derive the optimal policy

$$\pi = [3, 3, 2, 0], \quad (5.14)$$

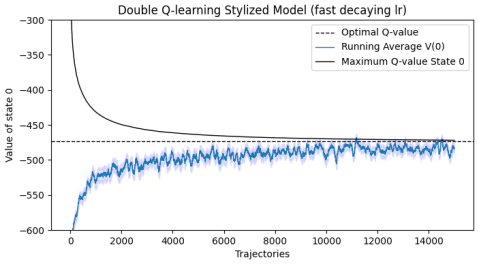
with corresponding values

$$\max_{a \in \mathcal{A}} Q^{\pi}(\cdot, a) = [-475.25, -469.14, -466.75, -464.62]. \tag{5.15}$$

Figure 5.2 gives a graphical representation of the learning. Here we motivate our choice for the learning rate, as mentioned in Section 5.2.1. Figure 5.2a visualizes the learning for the chosen learning rate $\alpha_n = \frac{1}{n^{3/5}}$. As one can see the progression of the learned Q-value in state 0 is not very smooth, but already after approximately 2000 simulated trajectories the learned Q-value in state 0 fluctuates around the optimal Q-value. Figure 5.2b visualizes the learning for learning rate $\alpha_n = \frac{1}{n}$. Here the progression of the learned Q-value is smooth, but it takes much longer to converge. The smaller step size causes the process to converge more slowly.



(a) Learning rate $\alpha_n = \frac{1}{n^{3/5}}$, where n is the number of simulated trajectories



(b) Learning rate $\alpha_n = \frac{1}{n}$, where n is the number of simulated trajectories

Figure 5.2: Visualizations convergence double Q-learning, stylized model. The black line represents the learned Q-value in state 0. The thick blue line is the running average of the simulated value in state 0 over the 100 most recent simulated trajectories, and the lighter blue area represents the 95% confidence bound.

5.4 Deep Q-learning

In Deep Q-learning a neural network is used to estimate the Q-function. The advantage of deep Q-learning compared to tabular or double Q-learning is that it can handle models with a larger state and action space. A disadvantage is that due to the use of a neural network there are no convergence guarantees, like we also saw for REINFORCE in Chapter 4.

Similar to tabular and double Q-learning the goal is to select actions that maximize the expected discounted sum of rewards. Reinforcement learning is known to be unstable when non-linear function approximation is used [34]. The instability in deep Q-learning is caused by correlations in the sequence of observations, small updates to the Q-function that could change the policy significantly, and the correlations between the Q-values and the target values [23]. The target values are used to compare the learned Q-values with, and will be defined later.

The correlations in the sequence of observations can be reduced by randomized experience replay. The agent stores all transitions (S_t, A_t, R_t, S_{t+1}) in its memory. Once enough experience is gathered the agent takes a random mini-batch of transitions and learns from this random subset of its memory. The correlations between the Q-values and the target values can be reduced by updating the Q-values continuously, whereas the target values are updated periodically. In [23]

these two adjustments are combined in a deep Q-network (DQN) agent. We have implemented the DQN agent following the methods in [23].

We will use slightly different notation here for the transitions. We write (S_n, A_n, R_n, S'_n) instead of (S_t, A_t, R_t, S_{t+1}), where S'_n is the subsequent state after taking action A_n in state S_n. The transitions in the mini-batch are sampled uniformly at random, thus we do not care for the exact time-step.

The DQN agent parameterizes an approximate value function $Q_{\boldsymbol{\theta}_t}(S_n, A_n)$ by means of a neural network, where $\boldsymbol{\theta}_t$ represents the weights of the neural network at time t. The updates of the Q-values are based on the aforementioned random subset of transitions, sampled uniformly at random from the decision makers memory.

The target values are defined as

$$Q_t^{\text{target}} = R_n + \gamma \max_{a \in \mathcal{A}} Q_{\theta_t^-}(S'_n, a), \tag{5.16}$$

where $Q_{\boldsymbol{\theta}_t^-}$ is a parameterized Q-function in terms of the weights $\boldsymbol{\theta}_t^-$ of a periodically updated neural network, to which we refer as the target network. Note that this is not the same neural network as is used to parameterize the value function $Q_{\boldsymbol{\theta}_t}$.

To perform the update the mean squared error (MSE) loss between the current Q-values, $Q_{\theta_t}(S_n, A_n)$, and the target Q-values, Q_t^{target}, of a random mini-batch from the decision makers memory is calculated. Here, γ is the discounting factor, $\boldsymbol{\theta}_t$ are the parameters of the Q-network at time t, and θ_t^- are the parameters of the periodically updated target network at time t.

After the loss is determined, the gradient of the loss is determined by back-propagation through the neural network. Then a gradient descent method, similar to the gradient descent method in REINFORCE, is used to update the Q-network parameters. The target network parameters are updated periodically with the Q-network parameters after a pre-specified number of steps and held fixed in between.

5.4.1 Implementation

The neural network used in DQN is similar to the neural network used in REINFORCE. The main difference is that in this neural network the Softmax function is not necessary. This is because the neural network has to output the Q-value of each action given the input state, instead of a probability distribution over the actions. We refer to this neural network as the DQN-network. The DQN-network takes the observed state as input and outputs the Q-values for each action in the observed state.

Algorithm 6 presents the main procedure of DQN. Before the procedure starts we set the target model equal to the randomly initialized Q-model. Within the procedure the Q-model is updated in every time-step as follows. First we take a random mini-batch of N quadruplets (S_n, A_n, R_n, S'_n) for $n = 1, 2, \ldots, N$. Then the MSE loss between the predicted Q-values of the states S_n and the target Q-values of the subsequent states S'_n is calculated as follows

$$\mathcal{L}\left(Q_{\boldsymbol{\theta}_t}, Q_t^{\text{target}}\right) = \frac{1}{N} \sum_{n=1}^{N} \left(Q_{\theta_t}(S_n, A_n) - (R_n + \gamma \max_{a \in \mathcal{A}} Q_{\theta_t^-}(S'_n, a)) \right)^2. \tag{5.17}$$

Next the gradient of the loss function is determined by back-propagation and the weights of the Q-model are updated using the Adam [19] gradient descent method.

Algorithm 6 Function Deep Q-learning

Input: The model environment and DQN-NETWORK
Output: The approximated Q-function

function DQN(environment)
  Set target model parameters equal to the Q-model parameters
  for each trajectory **do**
    Set start state S
    for each time-step in the trajectory **do**
      Select action A under policy derived from Q-model for state S (ε-greedy)
      Take action A, observe reward R and next state S'
      Take a random mini-batch from the memory
      Calculate loss according to Eq. (5.17)
      Determine gradients by back-propagation
      Update parameters of the Q-network via gradient descent
      $S \leftarrow S'$
    Decay ε
    Update target model parameters
  return total rewards per trajectory and final Q-function approximation.

At the end of each trajectory we decay ε linearly (until the minimum ε_{min} is reached), and we set the weights of the target model equal to the weights of the most recent Q-model.

From the final Q-function approximation one can determine the policy by taking the action with the highest Q-value in each state.

5.4.2 Example

To illustrate the performance of DQN we use the example from Section 3.4. The result is obtained by simulating 5000 trajectories, each of length 1000.

The final Q-function, where the rows and columns represent the states and actions respectively, is

$$Q^{\pi} = \begin{bmatrix} -470.15 & -467.61 & -464.64 & \mathbf{-463.34} \\ -463.30 & -462.01 & -460.39 & \mathbf{-459.42} \\ -456.44 & -456.41 & -456.15 & \mathbf{-455.50} \\ \mathbf{-454.56} & -455.87 & -456.84 & -456.56 \end{bmatrix}. \tag{5.18}$$

From this Q-function we derive policy

$$\pi = [3, 3, 3, 0], \tag{5.19}$$

with corresponding values

$$\max_{a \in \mathcal{A}} Q^{\pi}(\cdot, a) = [-463.34, -459.42, -455.50, -454.56]. \tag{5.20}$$

Note that this is not the optimal policy, but that the values per state are higher than the optimal values in Section 3.4. For state 0 this can easily be seen in Figure 5.3. The simulated value for state 0 is, however, lower than the optimal value. Note that the learned Q-value for state 0 is slightly higher than the simulated discounted sum of rewards in state 0 because of the ε-greedy action selection.

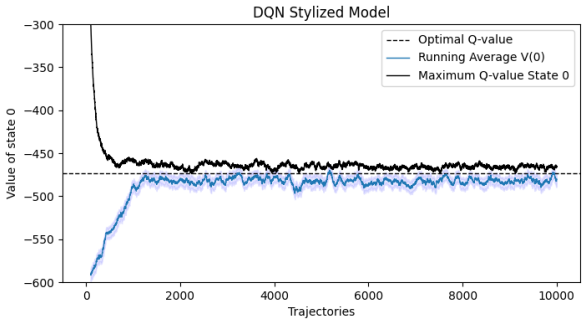


Figure 5.3: Visualization of the learning of the DQN algorithm, stylized model. The black line represents the learned Q-value in state 0. The thick blue line is the running average of the simulated value in state 0 over the 100 most recent simulated trajectories, and the lighter blue area represents the 95% confidence bound.

5.5 Concluding Remarks

In this chapter we presented three Q-learning methods. Tabular Q-learning is guaranteed to converge under the conditions discussed in Section 5.2.1, but might suffer from a maximization bias resulting in overestimated values. Double Q-learning could reduce this bias. The third method, DQN, should be able handle models with a larger state and action space, but there are no guarantees for convergence.

Table 5.1 summarizes the results for the three methods and Figure 5.4 visualizes the progression of the learned Q-value for state 0, i.e. $\max_{a \in \mathcal{A}} Q(0, a)$. Tabular and double Q-learning resulted in the optimal policy. In this example the maximization bias does not seem to occur for tabular Q-learning, as the learned Q-value approximately equals the exact $V^{\pi}(0)$ corresponding to the final learned policy. Double Q-learning, on the other hand, underestimates the value a little. DQN did not result in the optimal policy. The deviation of the value of state 0, $V^{\pi}(0)$, from the optimal value, $V^{*}(0)$, is -2.2%. In this example DQN overestimates the Q-values; the learned Q-value is higher than the optimal value.

Procedure	Learned Q	Exact $V^{\pi}(0)$	Deviation
Optimal DP	**−473.38**	**−473.38**	-
Q-learning	−473.58	−473.38	0%
Double Q-learning	−475.25	−473.38	0%
DQN	−463.34	−483.69	−2.2%

Table 5.1: Overview of the Q-learning results for the stylized model example in Section 3.4. Learned Q refers to the learned value of $\max_{a \in \mathcal{A}} Q(0, a)$. The exact $V^{\pi}(0)$ is the result from policy evaluation for the final policy π. The deviation represents the percentage deviation of the exact $V^{\pi}(0)$ from the optimal $V^{*}(0) = -473.38$.

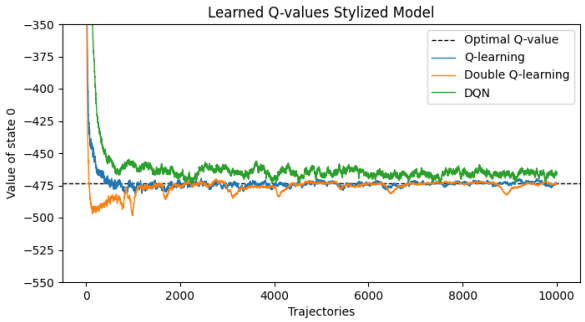


Figure 5.4: Learned Q-values for the stylized model example in Section 3.4.

Chapter 6

Data Analysis

Now that we have gathered insight into several RL algorithms for solving the single-item single-echelon inventory model, we proceed by finding realistic model parameters to investigate practical applicability. We do so by analyzing actual data from Philips's SPS network.

The data analysis consists of two parts. The first part is regarding the demand for several SKU's. Philips provided the SPS dataset for the DACH market (Germany, Austria, Switzerland). This dataset contains order line data of three years (2018, 2019, 2020). Table 6.1 contains a description of the columns of interest. The second part is regarding the transportation cost, for which we use the UPS rate structure data set as provided by Philips. This data set contains the prices to send packages per weight category.

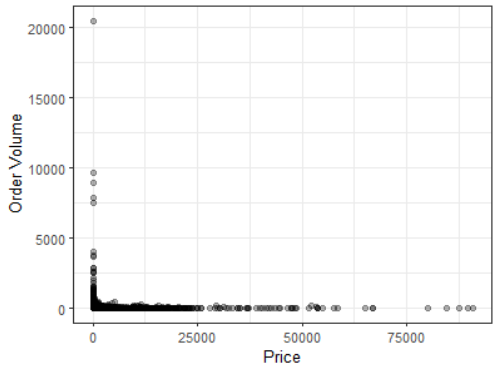
Column title	description
Part ID	Stock-keeping unit (SKU), unique identification number
Ship to	Location details to send the part to
Sales item date	Date and time the order was placed
Order quantity	Number of parts ordered
Standardprice	Price per part in Euro's
Total weight	Weight of the sent package, one package can contain multiple parts

Table 6.1: Description of the columns of interest in the SPS dataset.

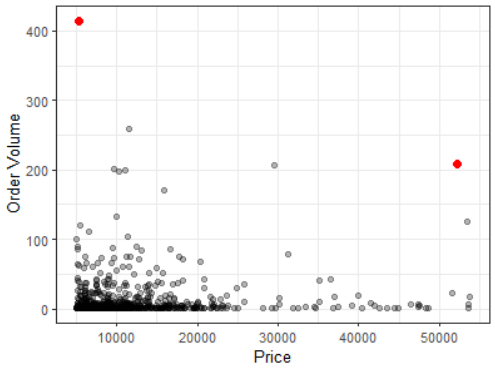
Since we are dealing with spare parts for medical systems, these can be very expensive parts, while requests are infrequent. Figure 6.1 provides an overview of the data in terms of the price versus the total order volume per item over three years. In Figure 6.1a the total order volume is plotted against the price per item for all items. The points in the top left represent inexpensive SKU's with a high order volume, whereas points in the bottom left represent expensive SKU's with a low order volume. From this plot one can see immediately that a majority of the parts has a very low order volume. The inexpensive SKU's with a high order volume can be cable ties for example, that are often needed for repairs.

Figure 6.1b shows the price versus the order volume for a zoom-in on the price range €5000 - €55.000. Figures 6.1c and 6.1d show similar plots as Figures 6.1a and 6.1b, but on a logarithmic scale, which provides more insight as the points are more widely distributed in the plots.

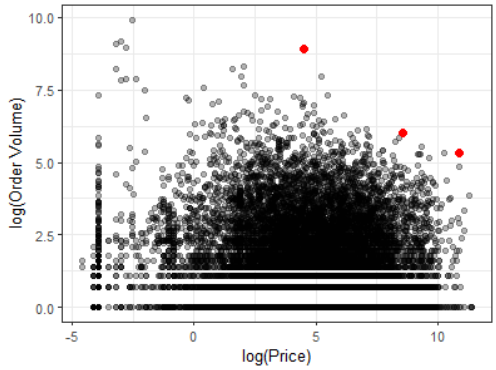
Based on the prices and order volumes, we have selected three SKU's to investigate further, these SKU's are highlighted in red in Figure 6.1. The selected SKU's all have a reasonable order volume of at least 200 in three years. The SKU's are chosen in three price categories of around €100, €5000, and €50.000. Note that Figures 6.1b and 6.1d contain two of the highlighted points, as the third point lies outside the zoom-in price range.



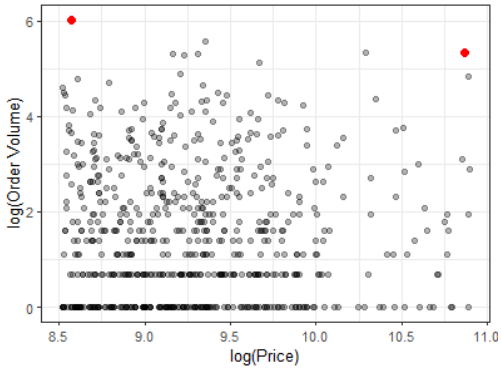
(a) Price versus order volume, all data



(b) Price versus order volume, prices limited to the range €5000 - €55.000



(c) Price versus order volume on logarithmic scale, all data



(d) Price versus order volume on logarithmic scale, prices limited to the range €5000 - €55.000

Figure 6.1: Scatterplots prices versus order volumes of SKU's in the SPS data set.

6.1 Demand Analysis

Table 6.2 contains the price in €, the weight in kg (used to determine the transportation cost in Section 6.2), the total number of placed orders and the total order volume for the three chosen SKU's. For SKU 1 and SKU 2 the order volume is higher than the total number of orders, which means that in some orders multiple parts were requested.

Before a comprehensive analysis, for the three SKU's separately, is executed we zoom into the data and clean the data if necessary. First notice that Philips operates in a next business-day delivery policy, so no demand is fulfilled on Saturday and Sunday. Table 6.3 gives an overview

SKU	Price (€)	Weight (kg)	Orders	Volume
1	90,67	0.20	5928	7557
2	5283,82	4.10	411	415
3	52201,35	117	208	208

Table 6.2: Price, weight, number of orders and total order volume over three years for the three chosen SKU's.

of the number of parts requested on each day of the week. For SKU 1 four parts were requested on a Saturday, but no other parts were requested during the weekend. Since four is a relatively small number, and taking into account the next business-day delivery policy, we assume that it is rare to receive orders during the weekend. Therefore we decide to exclude the weekend days from the data for the rest of the analysis, and our numerical experiments. One could say that we pretend that time stands still during the weekends, as nothing happens then.

Day	SKU 1	SKU 2	SKU 3
Mon	1381	78	45
Tue	1622	83	38
Wed	1557	87	41
Thu	1518	90	49
Fri	1468	77	34
Sat	4	0	0
Sun	0	0	0

Table 6.3: Daily order volume per SKU.

To give a first impression of the demand distribution for each SKU, Figure 6.2 shows histograms of the frequency of the daily order volume (weekend days excluded). The daily order volume is the summed demand per day, this can be multiple orders.

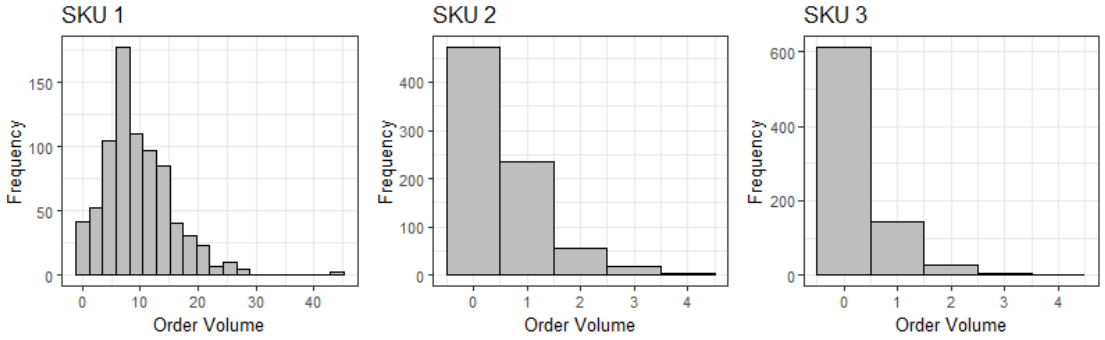


Figure 6.2: Histograms of the frequency of total order volume per day (weekend days excluded) for SKU 1, SKU 2 and SKU 3.

For SKU 1 we notice there are a few observations with a relatively high order volume. These might be outliers, so we further zoom in on these order volumes. A common way to detect outliers is by means of the inter quartile range (IQR), where observations that fall below $q_1 - 1.5 \cdot \text{IQR}$

or above $q_3 + 1.5 \cdot \text{IQR}$, where q_1 is the first and q_3 the third quartile, are classified as outliers. For SKU 1 we have $q_1 = 5$ and $q_3 = 13$, which results in upper bound 25 for the order volume. This method is based on the Gaussian distribution, which is a symmetric distribution. However, as one can see in Figure 6.2 the distribution of the order volume is right skewed. Therefore we adjust the upper bound to $q_3 + 2 \cdot \text{IQR}$ and thus allow higher order volumes. For SKU 1 this adjustment results in upper bound 29.

Based on the adjusted upper bounds we find two outliers, order volume 44 (on 29-04-2018) and order volume 43 (on 27-03-2020). We decide to remove the order volumes of 43 and 44 for SKU 1 from the dataset.

To provide a general overview of the order quantities per weekday, month and year we generated boxplots, see Figure 6.3. The first column visualizes the order quantity for SKU 1, the second column for SKU 2 and the third column for SKU 3. The first row contains boxplots per weekday, the second row contains boxplots per month, and the third row per year. For completeness we have not yet removed the outliers to generate these plots. The outliers will be removed before analyzing the demand distribution in Section 6.1.1. For SKU 1 we notice that there might be an effect of the weekday and month on the order quantity. However, we have decided not to further investigate this effect as the data is only used to provide a case for applicability of reinforcement learning.

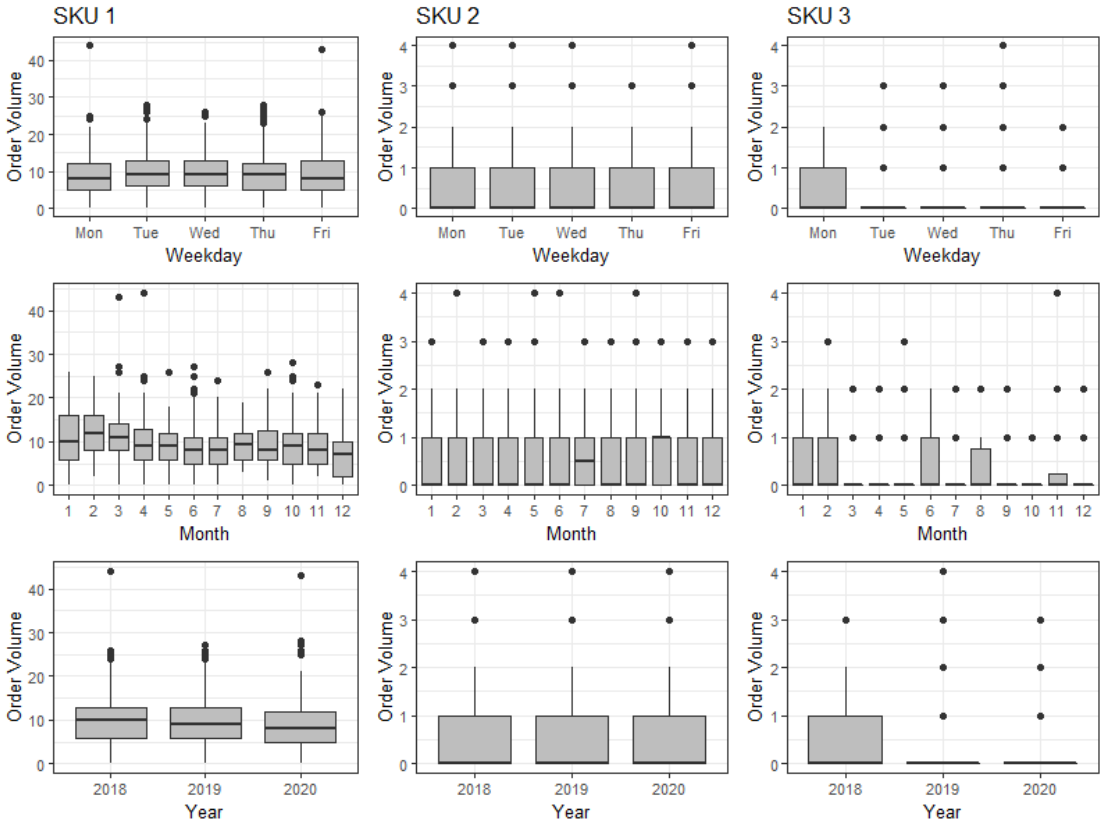


Figure 6.3: Boxplots of the order volume per weekday, month or year for the three parts, weekend days excluded.

If one would further investigate the effect of weekday, month or year, it would be possible to develop a more precise time-series demand model. A disadvantage here is that we only have data of three years. Therefore it is harder to generalize the predicted order quantities. In terms of the state space it would also complicate things. Besides keeping track of the inventory level, we would also need to keep track of the date, in order to be able to predict the demand. For practical purposes we would like to keep things simple by fitting one stationary distribution per SKU without taking into account any effect of the year, month or day of the week.

6.1.1 Distribution Fitting

Before we start fitting distributions, we provide some summary statistics per SKU in Table 6.4. Since we are dealing with order volumes we consider discrete distributions with non-negative support. We consider the Poisson, geometric, and negative binomial distribution.

SKU	Mean	Var	Min	Max
1	9.538	31.194	0	28
2	0.529	0.584	0	4
3	0.264	0.297	0	4

Table 6.4: Mean, standard deviation, minimum and maximum of the daily order volume per SKU (data 2018, 2019 and 2020 all together, outliers excluded).

We estimate the parameters of each distribution using the method of moments. To ease the notation we write $\mu = \mathbb{E}[D]$ and $\sigma^2 = \text{Var}(D)$, where D is the demand distribution. For the Poisson distribution with parameter λ we have

$$\lambda = \mu. \tag{6.1}$$

For the geometric distribution with parameter p we have

$$p = \frac{1}{1+\mu}. \tag{6.2}$$

For the negative binomial distribution with parameters n and p we have

$$n = \frac{\mu^2}{\sigma^2 - \mu} \quad \text{and} \quad p = \frac{\mu}{\sigma^2}. \tag{6.3}$$

Note that the Poisson and geometric distribution only have one parameter, whereas the negative binomial distribution has two parameters. Therefore the negative binomial distribution has more flexibility in fitting the distribution and can account for overdispersion. Overdispersion means there is more variability in the distribution than predicted by the fitted distribution. For example, for the theoretical Poisson distribution we have that both the expectation and the variance equal λ. In Table 6.4 we see that for all three SKU's the variance is greater than the mean. Therefore we would predict less variability using the Poisson distribution than there actually is in the data.

Next we determine the parameters for each distribution per SKU and provide QQ-plots to provide insight in the best fit. We will not use a goodness-of-fit test, as such tests tend to become biased if one is to use the same dataset to estimate the parameters and run the test [15]. A second issue that needs to be addressed is that the sample size should be large enough when applying

goodness-of-fit tests, see [21] for more details. Furthermore, a key assumption for most goodness-of-fit tests, like for the Kolmogorov-Smirnov test, is that the target CDF is continuous. For the target distributions we consider this is obviously not the case, thus the continuity assumption is violated. Due to time limitation we decided not to go into detail on the distribution fitting using goodness-of-fit tests, but leave it as a topic for future research.

Remark. *Note from Figure 6.3 that the support of the order volume for SKU 2 and 3 consists of only five points,* $\{0, 1, 2, 3, 4\}$. *Due to the limited support it is challenging to fit a distribution for these SKU's. Better distribution fitting is left as a topic for future research.*

SKU 1

For SKU 1 the parameter estimates for the Poisson, geometric and negative binomial distribution can be found in Table 6.5.

Distribution	Parameter estimate
Poisson	$\lambda = 9.538$
Geometric	$p = 0.095$
Negative binomial	$n = 4.201$
	$p = 0.306$

Table 6.5: Parameter estimation for several distributions, SKU 1.

We use these parameters to generate QQ-plots to determine whether or not the demand distribution is similar to the theoretical distributions with estimated parameters. If the points in the QQ-plot approximately lie on the blue reference line (order volume quantiles = theoretical quantiles), then the demand approximately follows the theoretical distribution. In Figure 6.4 we see that this is the case for the negative binomial distribution, but not for the Poisson and geometric distribution. Therefore we decide to use the negative binomial distribution with parameters $n = 4.201$ and $p = 0.306$ as demand distribution for SKU 1.

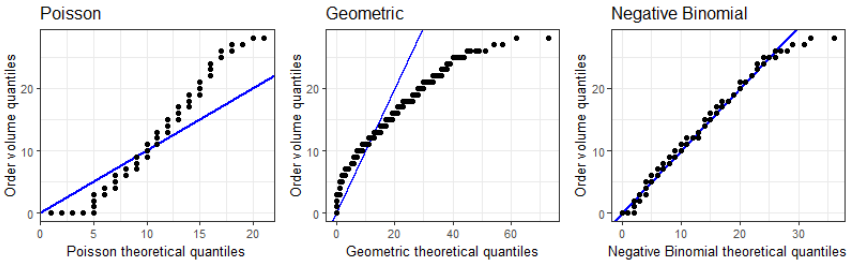


Figure 6.4: QQ-plots to compare the demand distribution of SKU 1 with the theoretical Poisson, geometric and negative binomial distribution.

SKU 2

For SKU 2 the parameter estimates for the Poisson, geometric and negative binomial distribution can be found in Table 6.6.

Distribution	Parameter estimate
Poisson	$\lambda = 0.529$
Geometric	$p = 0.654$
Negative binomial	$r = 5.120$
	$p = 0.906$

Table 6.6: Parameter estimation for several distributions, SKU 2.

Figure 6.5 shows the QQ-plots for SKU 2. Here it is not immediately clear which distribution has the best fit. In fact one might argue that neither of the three distributions fits well. For all three distributions, some point lie on the blue reference line, but there are also points off the line. For the geometric distribution most points lie below the reference line, therefore we eliminate the geometric distribution. The Poisson and negative binomial distribution have approximately the same amount of points on both sides of the reference line, but the negative binomial distribution seems to be more balanced.

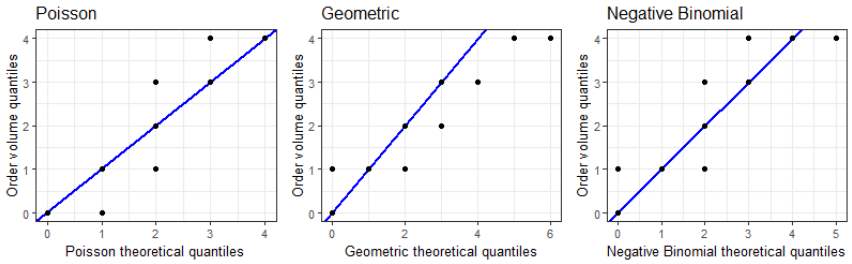


Figure 6.5: QQ-plots to compare the demand distribution of SKU 2 with the theoretical Poisson, geometric and negative binomial distribution.

Before we decide on the demand distribution we also take a look at the robustness of the policy under the demand distribution. The optimal policies for the cost parameters as discussed in the next section for the Poisson and negative binomial demand distribution are provided in Table 6.7.

Looking at the policies we only see a small difference in state 2. Looking at the values of state 0 for each demand distribution we see that the deviation is within a range of 5%. From this we conclude that the policy is reasonably robust under these two demand distributions. We decide to use the negative binomial distribution with parameters $n = 5.120$ and $p = 0.906$ as demand distribution for SKU 2. If the demand distribution in fact would follow a Poisson distribution instead, we underestimate the expected long term rewards since the value in state 0 is a bit lower for the negative binomial distribution than for the Poisson distribution. From a managerial perspective this is better than overestimating the rewards, as it is better to have higher rewards than expected compared to lower rewards than expected.

Demand distribution	π	$V^\pi(0)$	5% deviation range
Poisson	$[4, 3, 2, 0, 0, 0]$	-689.32	$(-723.79, -654.85)$
Negative binomial	$[4, 3, 3, 0, 0, 0]$	-709.25	$(-744.71, -673.79)$

Table 6.7: Optimal policy and value in state 0 for a Poisson and negative binomial demand distribution for SKU 2.

SKU 3

For SKU 3 the parameter estimates for the Poisson, geometric and negative binomial distribution can be found in Table 6.8.

Distribution	Parameter estimate
Poisson	$\lambda = 0.264$
Geometric	$p = 0.791$
Negative binomial	$n = 2.131$
	$p = 0.890$

Table 6.8: Parameter estimation for several distributions, SKU 3.

Figure 6.6 shows the QQ-plots for SKU 3. For the negative binomial distribution all points lie on the blue reference line. Therefore we decide to use the negative binomial distribution with parameters $n = 2.131$ and $p = 0.890$ as demand distribution for SKU 3.

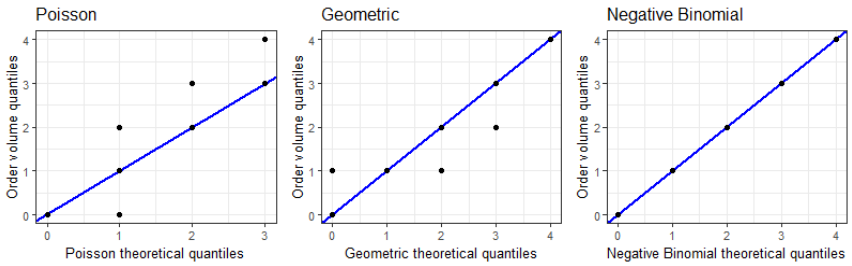


Figure 6.6: QQ-plots to compare the demand distribution of SKU 3 with the theoretical Poisson, geometric and negative binomial distribution.

We conclude this section with a visualization of the fitted demand distributions for each SKU. Figure 6.7 shows the histograms of the frequency of total order volume per day, per SKU. The fitted demand distributions were scaled and added to the frequency histograms to visualize the fit.

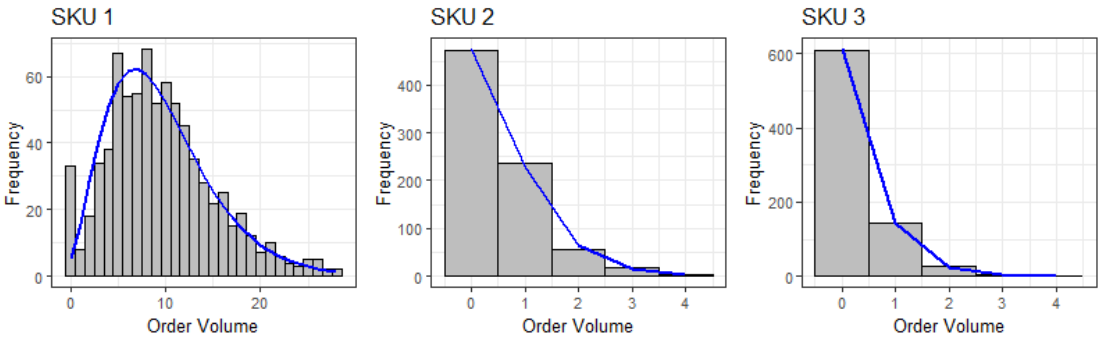


Figure 6.7: Histograms of the frequency of total order volume per day, per SKU, with corresponding fitted demand distribution. The distribution is scaled to the frequency.

6.2 Cost Analysis

As described in Chapter 3 the involved cost are holding cost, transportation cost and emergency cost. Table 6.9 contains the cost parameters for SKU 1, 2 and 3.

The holding cost per year is 10% of the price of an SKU. To obtain the holding cost h per day we divide this amount by 365 days.

The transportation cost is a combination of fixed and variable cost per part. The UPS rate structure data set contains prices per package for weight categories. One package can contain multiple parts. We performed a linear regression analysis to fit the actual transportation costs to the linear transportation cost structure used in our model, see Appendix D for the details. The prices and weights for each SKU are presented in Table 6.2.

The emergency cost is tuned in order to achieve a part availability of 95%, the mentioned target in Section 1.1. We will elaborate on this tuning in the next subsection.

Type of cost		SKU 1	SKU 2	SKU 3
Holding cost	h	0.02	1.45	14.30
Fixed transportation cost	τ_f	3.15	7.73	0
Variable transportation cost	τ_v	0.28	0.93	42.12
Emergency cost	e	0.60	50	900
Warehouse capacity	$s_{max} = a_{max}$	70	5	5

Table 6.9: Cost and capacity parameters per SKU, all cost in €.

6.2.1 Emergency Cost Tuning

The emergency cost is used to indirectly comply with the part availability target of 95% per SKU. The tuning is done by trial and error using a simulation, where we aim for part availability rate ≥ 0.95. In the simulation we keep track of the total demand, and the total number of parts that could not be delivered from the LDC, i.e. the excess demand. The part availability rate is then easily determined as $1 - \frac{\text{total excess demand}}{\text{total demand}}$.

	Emergency cost	Availability rate	CI
SKU 1	0.5	0.9273	$(0.9273, 0.9274)$
	0.6	0.9523	$(0.9523, 0.9524)$
SKU 2	40	0.9164	$(0.9162, 0.9167)$
	50	0.9721	$(0.9720, 0.9722)$
SKU 3	800	0.9208	$(0.9204, 0.9211)$
	900	0.9815	$(0.9813, 0.9816)$

Table 6.10: Part availability rates with corresponding 95% confidence intervals for two penalties per SKU.

Note that the optimal policy can change when the emergency cost changes. When the policy remains the same for different emergency cost values, the availability rate will also remain the same on average. For SKU 2 the policy and availability rate remain the same for emergency cost

$e \in [40, 45]$. The smallest value for which an availability rate of at least 0.95 is reached, is $e = 46$. We choose to round the emergency cost to $e = 50$, as the policy and availability rate are the same as for $e = 46$. To reach an availability rate of at least 0.95 for SKU 3, the smallest value is $e = 893$. For $e \in [800, 892]$ the policy and availability rate are the same. We choose to round the penalty value to $e = 900$, as the policy and availability rate are the same as for $e = 893$.

Here we state the optimal policies resulting from dynamic programming for the parameter values in Table 6.9.

$$\pi_1 = [58, 58, 58, 58, 58, 58, 58, 57, 57, 56, 56, 55, 54, 54, 53, 52, 51, 0, 0, \ldots, 0] \tag{6.4}$$

$$\pi_2 = [4, 3, 3, 0, 0, 0] \tag{6.5}$$

$$\pi_3 = [2, 1, 1, 0, 0, 0] \tag{6.6}$$

Chapter 7

Results Stylized Model

Now that we have found realistic model parameters for the single-item single-echelon inventory model at Philips Healthcare, we will apply the RL approaches. In this chapter the results for the three SKU's from the previous chapter are provided. For each SKU the optimal policy is obtained using dynamic programming and the REINFORCE, tabular Q-learning, double Q-learning and DQN procedures are performed. Per SKU the results from the RL approaches will be compared with the optimal policy.

The results will be compared based on both the learned policy and the corresponding value of state 0, $V^{\pi}(0)$, where π refers to the learned policy at the end of the procedure. The Q-learning methods all output a deterministic policy. For REINFORCE we translate the resulting stochastic policy to the approximated deterministic policy by setting $\pi(s) = \max_{a \in \mathcal{A}} \pi(a \mid s)$. The policies are visualized using heat maps, that way one can easily see the differences between the policies in terms of which action to take in which state. $V^{\pi}(0)$ is used as a performance measure, as it is intuitive to start with an empty LDC and from there apply the learned policy.

Note that it is a bit unfair to compare REINFORCE with the Q-learning methods based on the simulated values, since the Q-learning methods use ε-greedy action selection with a minimum value $\varepsilon_{min} = 0.01$ and therefore continue to act in a sub-optimal manner during the learning process. We will therefore determine the exact $V^{\pi}(0)$ for the resulting policy for each algorithm using policy evaluation and compare that value with the optimal value $V^{*}(0)$.

7.1 SKU 1

The final policies resulting from each of the methods considered in this report for SKU 1 are visualized in Figure 7.1. Note that this visualization only contains the first 26 states to keep the figure readable. For the complete policies we refer to Appendix E.

It immediately stands out that both tabular Q-learning and double Q-learning seem to have learned a rather odd policy. One would expect a monotonically decreasing policy, but for tabular and double Q-learning this is clearly not the case. Looking at Figure 7.2b, we see that the learned Q-values have not converged yet after 10.000 trajectories. In fact, based on Figure 7.2b, we suspect that the process needs way more trajectories to converge. The learned Q-values for tabular and double Q-learning in Table 7.1 are the results after 100.000 trajectories, see Appendix E for the corresponding figure. These values are still far off from the optimal value.

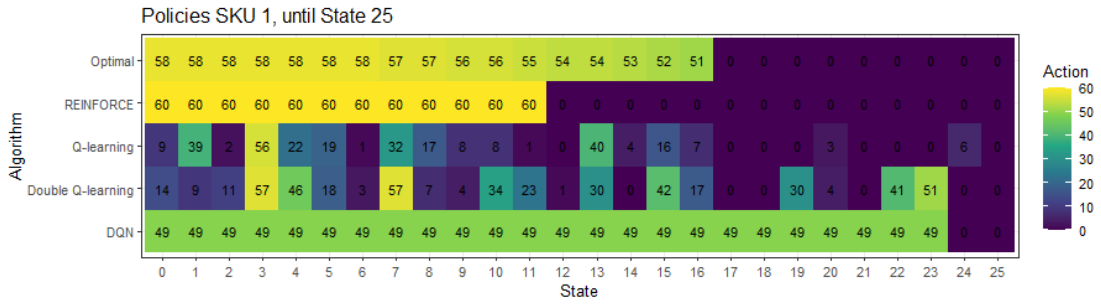
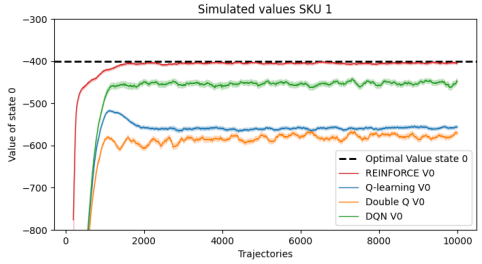


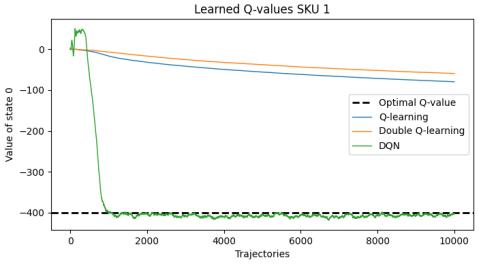
Figure 7.1: The learned policies for states 0-25 for SKU 1.

REINFORCE and DQN both seem to have learned a better, but not the optimal policy. REINFORCE resulted in a policy with fixed reorder point 11 and fixed order quantity 60, i.e. if there are ≤ 11 parts available in the LDC, order 60 parts at the RDC. DQN resulted in a similar policy, but with reorder point 23 and fixed order quantity 49. As one can see in Table 7.1 the corresponding exact value in state 0 for the REINFORCE policy is slightly higher than for the DQN policy, but both do not deviate much from the optimal value.

Although the exact values for REINFORCE and DQN are relatively close, the simulated values for state 0 are not, as one can see in Figure 7.2a. The difference in simulated values is (partially) caused by the sub-optimal behavior of DQN due to the ε-greedy action selection.



(a) Running average of 200 most recent simulated values, i.e. discounted sum of rewards from start state 0, for SKU 1. The light shaded areas represent the 95% confidence bounds for the running average.



(b) Learned Q-values for SKU 1.

Figure 7.2: Progression of the simulated $V(0)$ and learned Q-values for SKU 1.

Remark. *In the setting as discussed in Chapter 4 the REINFORCE algorithm did not work for SKU 1. The neural network estimating the policy was returning NaN probabilities. This problem is probably caused by the structure of the neural network. We suspect that the Softmax function is causing trouble due to numerical instability. We slightly modified the input to be a unit vector representing the current inventory level instead of a single number, after which the REINFORCE algorithm did work. It is worth noting that for a smaller state space, i.e.* $s_{max} = 30$ *this problem did not occur in the original setting.*

Procedure	Learned Q	Exact $V^\pi(0)$	Deviation
Optimal DP	**−399.98**	**−399.98**	-
REINFORCE	-	−404.37	−1.1%
Q-learning	−201.55	−590.97	−47.7%
Double Q-learning	−152.98	−533.28	−33.3%
DQN	−403.06	−405.90	−1.5%

Table 7.1: The results for each method for SKU 1. Learned Q refers to the learned value of $\max_{a \in \mathcal{A}} Q(0, a)$. The exact $V^\pi(0)$ is the result from policy evaluation for the final policy π. The deviation represents the percentage deviation of the exact $V^\pi(0)$ from the optimal $V^*(0) = -399.98$.

7.2 SKU 2

For SKU 2 the resulting policies are visualized in Figure 7.3. Here the tabular Q-learning procedure resulted in the optimal policy, the other methods did not. Looking at the exact $V^\pi(0)$ corresponding to the final policies, the double Q-learning algorithm resulted in the second best policy. The exact $V^\pi(0)$ for the policy learned by double Q-learning deviates only 0.4% from the optimal value in state 0. Note that tabular Q-learning slightly overestimates the learned Q-value, as mentioned in Chapter 5 this is the maximization bias. Double Q-learning does not suffer from this bias, the learned Q-value is approximately equal to the exact $V^\pi(0)$.

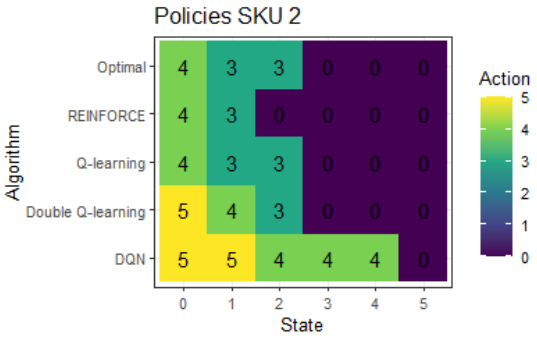
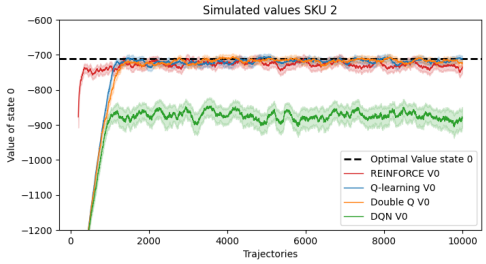


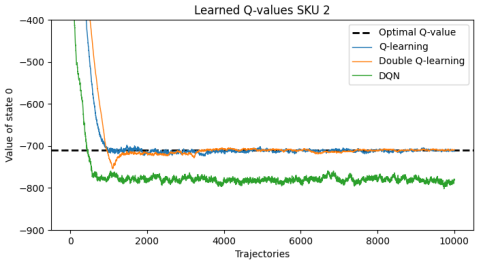
Figure 7.3: Learned policies for SKU 2.

Although REINFORCE did not learn the optimal policy, the exact $V^\pi(0)$ corresponding to the learned policy only deviates 1.9% from the optimal value. With the sub-optimal behavior of tabular and double Q-learning caused by ε-greedy action selection, the simulated values for REINFORCE, tabular and double Q-learning are all around the same value, as can be seen in Figure 7.4a.

The DQN algorithm is performing very poorly. For SKU 2 DQN overestimates the policy as for five of the six states the action is higher than in the optimal policy. It is remarkable that the exact $V^\pi(0)$ is much worse than the learned Q-value. The exact $V^\pi(0)$ deviates 34.7% from the optimal value, whereas the learned Q-value deviates 10.1% from the optimal policy. It is unclear what causes this difference.



(a) Running average of 200 most recent simulated values, i.e. discounted sum of rewards from start state 0, for SKU 2. The light shaded areas represent the 95% confidence bounds for the running average.



(b) Learned Q-values for SKU 2.

Figure 7.4: Progression of the simulated $V(0)$ and learned Q-values for SKU 2.

Procedure	Learned Q	Exact $V^\pi(0)$	Deviation
Optimal DP	**−710.08**	**−710.08**	-
REINFORCE	-	−723.52	−1.9%
Q-learning	−708.07	−710.08	0
Double Q-learning	−712.30	−712.61	−0.4%
DQN	−781.45	−956.76	−34.7%

Table 7.2: The results for each procedure for SKU 2. Learned Q refers to the learned value of $\max_{a \in \mathcal{A}} Q(0, a)$. The exact $V^\pi(0)$ is the result from policy evaluation for the final policy π. The deviation represents the percentage deviation of the exact $V^\pi(0)$ from the optimal $V^*(0) = -710.08$.

7.3 SKU 3

Figure 7.5 represents the learned policies for SKU 3. Again only tabular Q-learning resulted in the optimal policy and double Q-learning resulted in the second best policy with value $V^\pi(0)$ deviating only 0.3% from the optimal value $V^*(0)$. Here we do not see an overestimation of the learned Q-value for tabular Q-learning.

REINFORCE again learned a sub-optimal policy, with a small deviation from the optimal value of 1.0%. REINFORCE has learned this policy very quickly; already after approximately 1000 simulated trajectories as one can see in Figure 7.6a. In Figure 7.6 we see that for tabular and double Q-learning the simulated values and the learned Q-values stabilize after approximately 2000 trajectories. Again we observe that the sub-optimal behavior of REINFORCE, tabular and double Q-learning lead to similar simulated values for state 0.

DQN is again the worst performing algorithm, but the deviation of the exact $V^\pi(0)$ from the optimal $V^*(0)$ is not as bad as it is for SKU 2. The learned Q-value is however worse than the exact $V^\pi(0)$, contrary to what we saw for SKU 2.

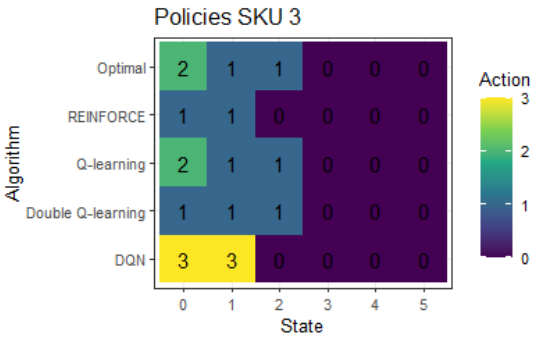
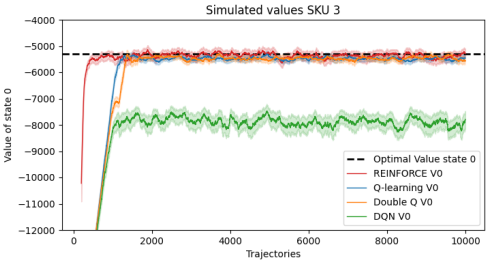
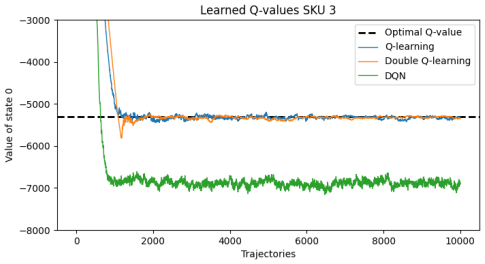


Figure 7.5: Learned policies for SKU 3.



(a) Running average of 200 most recent simulated values, i.e. discounted sum of rewards from start state 0, for SKU 3. The light shaded areas represent the 95% confidence bounds for the running average.



(b) Learned Q-values for SKU 3.

Figure 7.6: Progression of the simulated $V(0)$ and learned Q-values for SKU 3.

Procedure	Learned Q	Exact $V^\pi(0)$	Deviation
Optimal DP	**−5309.34**	**−5309.34**	-
REINFORCE	-	−5363.53	−1.0%
Q-learning	−5320.80	−5309.34	0
Double Q-learning	−5329.57	−5325.14	−0.3%
DQN	−6879.06	−5745.16	−8.2%

Table 7.3: The results for each procedure for SKU 3. Learned Q refers to the learned value of $\max_{a\in\mathcal{A}} Q(0,a)$. The exact $V^\pi(0)$ is the result from policy evaluation for the final policy π. The deviation represents the percentage deviation of the exact $V^\pi(0)$ from the optimal $V^*(0) = -5309.34$.

7.4 Concluding Remarks

Now that we have presented the results per SKU, we will give some concluding remarks regarding the RL methods applied to the Philips case. The tabular Q-learning was the only method that learned the optimal policy for two of the three SKU's. If one has more time (or computational power) the tabular Q-learning should result in the optimal policy eventually, even for SKU 1, because of the guaranteed convergence as discussed in Chapter 5. The other methods did not learn the optimal policy for any of the three SKU's.

For SKU 2 we observed a slight overestimation of the Q-value, but for SKU 3 we did not.

From the results we see that, although REINFORCE did not learn the optimal policy, the exact value $V^{\pi}(0)$ only deviates 1-2% from the optimal value $V^{*}(0)$ for each SKU. In all three cases REINFORCE had learned the sub-optimal policy after approximately 2000 simulated trajectories, which takes approximately 13 minutes. Based on these results REINFORCE is the most consistent algorithm.

The DQN method is the least consistent and worst performing method considering the three cases. There is no clear pattern in the behavior of DQN. For SKU 1 it performs quite well. For SKU 2 it overestimates the policy and the exact $V^{\pi}(0)$ is way worse than the learned Q-value, but for SKU 3 this is not the case. Due to time limitations we did not investigate the DQN method as thoroughly as the REINFORCE, tabular and double Q-learning methods. More research on this method is needed in order to better understand and explain this behavior.

It is striking that for SKU 1, the one case with a large state and action space, the DRL methods REINFORCE and DQN outperform tabular and double Q-learning, whereas for SKU 2 and 3 this is the other way around. On the other hand we see a distinction between the DRL approaches, with REINFORCE performing quite well in all three situations, whereas DQN does not. It could be that the value-based neural network is more vulnerable to the non-convexity than the policy-based neural network. It could also be the case that DQN may not be suitable for small problems or that the structure of the neural network is simply not suitable. Further research is needed to gain a better understanding of the performance of DQN and the DRL methods in general.

All results presented above pertain to the stylized model. Philips' service part supply chain is however much more complex. This motivated us to formulate an extension of the stylized model as described in the next chapter.

Chapter 8

A Model Extension

Up until this point we have considered the stylized model. However, it is not very realistic to assume there is only one LDC, as Philips' service part supply chain is more complex. Therefore we propose an extension of the stylized model, in which we have two LDC's instead of one. We will start this chapter explaining the model structure.

Figure 8.1 gives a schematic representation of the structure. Both LDC's are replenished by the RDC, which is still assumed to have infinite stock. Each LDC has its own customer group from which demand occurs. The demand from customer group i is immediately met if LDC i has enough parts available. The black arrows indicate the regular replenishment and delivery to customers. If LDC 1 does not have enough parts available to meet the demand from customer group 1, LDC 2 can deliver parts to customer group 1 as well at additional cost (to which we will refer as penalty cost) and vice versa. These shipments are indicated by the blue arrows in Figure 8.1. Note that both LDC's will first meet the demand from their own customer group before sending parts to the other customer group. If LDC 1 and LDC 2 together do not have enough parts available to meet the demand from customer group 1 and 2 together, the demand will be met by emergency shipments from the RDC at additional cost (to which we will refer to as emergency cost). In Figure 8.1 this is indicated by the dashed orange arrows.

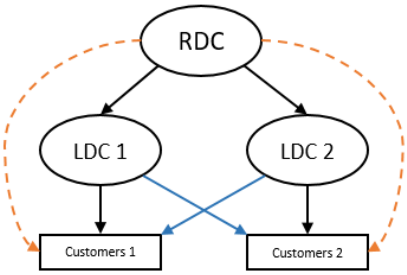


Figure 8.1: Model extension flow chart.

Although the complexity of the model increased by adding an extra LDC, this model can still be formulated as an MDP. A mathematical formulation will be given in Section 8.1. We will, however, not provide closed formulas for the transition probabilities and expected rewards in general, like we did for the stylized model.

Due to time constraints we choose not to apply all RL approaches as discussed in this thesis to the model extension, but only Q-learning because of the convergence guarantees. Since we do not have a closed formula for the transition probabilities and the expected rewards, we do not have a baseline to compare the results with. To validate the Q-learning method for the model extension we provide various intuitive examples in Section 8.2. For one of the examples we determined the optimal solution, so that we can compare the Q-learning result with the optimal result.

8.1 Finite Markov Decision Process

In the extended model we duplicate the stylized model and add interaction between the LDC's and the customer groups. The notation will be similar to the stylized model in Section 3.1.

The assumptions for the model are similar as the assumptions for the stylized model. Here we state three additional assumptions.

1. The customer groups are independent. Per group the customer demand distribution is i.i.d and stationary.
2. If the demand of a customer group exceeds the inventory level of the corresponding LDC, the other LDC will meet (part of) the excess demand at additional penalty cost. If (part of) the excess demand cannot be met by the other LDC, there will be an emergency shipment from the RDC at additional emergency cost.
3. LDC i has a maximum capacity of s^i_{max} parts, for $i \in \{1, 2\}$.

Let $S_t = (I^1_t, I^2_t)$ be the state representing the inventory level at LDC 1 and LDC 2 at time t. The spare parts ordered at time t at the RDC, denoted by $a_t = (a^1_t, a^2_t)$, arrive just before time $t+1$, like in the stylized model. Note that I^i_t is the inventory level at LDC i, and a^i_t is the number of spare parts ordered for LDC i at time t. The stochastic customer demand $D_t = (D^1_t, D^2_t)$, where D^i_t is the demand from customer group i, occurs right after the decision on a_t is made. Note that the customer demand is stationary, the time indication is used for generality. D^1_t and D^2_t are independent, but not necessarily identically distributed. To ease the notation we write $\widehat{I}^i_t = (I^i_t - D^i_t)^+$ for the inventory surplus and $\widehat{D}^i_t = (D^i_t - I^i_t)^+$ for the excess demand at LDC i at time t for $i \in \{1, 2\}$. The inventory level transitions can now be defined as

$$I^1_{t+1} = \min\left\{\max\{0, \widehat{I}^1_t - \widehat{D}^2_t\} + a^1_t, s^1_{max}\right\}, \tag{8.1}$$

$$I^2_{t+1} = \min\left\{\max\{0, \widehat{I}^2_t - \widehat{D}^1_t\} + a^2_t, s^2_{max}\right\}. \tag{8.2}$$

The cost structure can be defined in a similar fashion as for the stylized model. The holding cost, transportation cost and emergency cost for both LDC's are defined as in Section 3.1, where we add a superscript $i \in \{1, 2\}$ to indicate the corresponding LDC as a duplicate of the stylized model.

We define penalty cost related to the interaction between LDC's and customer groups. Without this interaction the model can be split up in two independent models, each similar to the stylized model. The penalty cost to deliver x spare parts from LDC i to the other customer group is represented by $P^i(x) = \rho^i \cdot x$, where $\rho^i \geq 0$ is the penalty per part. If $\rho^1 = \rho^2 = 0$

and $e^1 = e^2$ the model is equivalent to the stylized model with LDC capacity $s^1_{max} = s^2_{max}$ and customer demand $\tilde{D} = D^1 + D^2$.

The reward depends on the state $S_t = (I^1_t, I^2_t)$, the action $a_t = (a^1_t, a^2_t)$, and the demand $D_t = (D^1_t, D^2_t)$.

$$\begin{aligned} R_t(S_t, a_t, D_t) = &-H^1\left((\widehat{I}^1_t - \widehat{D}^2_t)^+\right) - H^2\left((\widehat{I}^2_t - \widehat{D}^1_t)^+\right) \\ &- P^1\left(\min\{\widehat{I}^1_t, \widehat{D}^2_t\}\right) - P^2\left(\min\{\widehat{I}^2_t, \widehat{D}^1_t\}\right) \\ &- E^1\left((\widehat{D}^1_t - \widehat{I}^2_t)^+\right) - E^2\left((\widehat{D}^2_t - \widehat{I}^1_t)^+\right) \\ &- O^1\left(a^1_t\right) - O^2\left(a^2_t\right). \end{aligned} \tag{8.3}$$

The holding cost in both LDC's are assumed to be equal $h_1 = h_2 = h$. Also the fixed ordering cost and variable ordering cost are assumed to be equal for both LDC's, i.e. $\tau^1_v = \tau^2_v = \tau_v$ and $\tau^1_f = \tau^2_f = \tau_f$.

In summary the model can be defined as follows.

Decision epochs:

$$\mathcal{T} = \{1, 2, \dots\}. \tag{8.4}$$

State space: the inventory level at LDC 1 and LDC 2 at the beginning of each time period

$$\mathcal{S} = \{0, 1, \dots s^1_{max}\} \times \{0, 1, \dots s^2_{max}\}. \tag{8.5}$$

Action space: the number of spare parts ordered at the RDC at the beginning of each time period, transported to LDC 1 and LDC 2 during each time period

$$\mathcal{A} = \{0, 1, \dots a^1_{max}\} \times \{0, 1, \dots a^2_{max}\}, \tag{8.6}$$

where we assume $a^i_{max} \leq s^i_{max}$ for $i \in \{1, 2\}$.

Rewards: one step reward for $t = 1, 2, \dots$

$$\begin{aligned} R(S_t, A_t, D_t) = &- h(I^1_t + I^2_t - D^1_t - D^2_t)^+ \\ &- \rho^1 \min\{\widehat{I}^1_t, \widehat{D}^2_t\} - \rho^2 \min\{\widehat{I}^2_t, \widehat{D}^1_t\} \\ &- e^1(\widehat{D}^2_t - \widehat{I}^1_t)^+ - e^2(\widehat{D}^1_t - \widehat{I}^2_t)^+ \\ &- O(a^1_t) - O(a^1_t). \end{aligned} \tag{8.7}$$

Transition probabilities: the probability to go from state $S_t = s \in \mathcal{S}$ to state $S_{t+1} = j \in \mathcal{S}$ when taking action $a_t = a \in \mathcal{A}$ is denoted by

$$p^a(s, j) = \mathbb{P}(S_{t+1} = j \mid S_t = s, a_t = a). \tag{8.8}$$

The optimality criterion is defined in Section 3.1.1.

8.2 Validation

We consider four cases to validate the model extension. The parameter settings for each of the four cases are provided in Table 8.1.

	Case 1	**Case 2**	**Case 3**	**Case 4**
D^1	Pois(1)	Pois(1)	Pois(1)	Pois(1)
D^2	Pois(1)	**0**	Pois(1)	Pois(1)
s^1_{max}	5	3	2	3
s^2_{max}	5	**0**	2	3
a^1_{max}	5	3	1	3
a^2_{max}	5	0	1	3
h	5	1	1	0
ρ^1	30	0	2	0
ρ^2	20	0	2	0
e^1	**0**	10	5	**500**
e^2	**0**	10	5	**500**
τ_v	5	1	1	0
τ_f	2	2	1	0

Table 8.1: Parameter settings for the validation cases for the model extension.

Case 1

In the first case we set both emergency cost equal to 0. The optimal policy is then to take action $(0, 0)$ in each state. This policy will result in two empty LDC's and all demand will be met via emergency shipments, without any cost. The other cost parameters can be set arbitrarily. The parameters we used are provided in Table 8.1. We simulated 5000 trajectories, after which the Q-learning procedure resulted in the policy where action $(0, 0)$ is taken in each state. The result is thus as expected.

Case 2

In case 2 we set the parameters such that the system will behave as the stylized model. We set the maximum capacity at LDC 2 and the customer demand from customer group 2 both equal to 0. The parameters are provided in Table 8.1. The model extension will behave as the stylized model example in Section 3.4, as there will only be customer demand from one group and LDC 2 cannot be used due to 0 capacity. We simulated 5000 trajectories for this experiment, after which the Q-learning procedure resulted in the optimal policy for the example in Section 3.4.

Case 3

In case 3 we set symmetric parameters. We expect the resulting policy to be symmetric as well. The parameters are provided in Table 8.1. For this case we choose a small state and action space. Because of the small size of the problem we were able to calculate the transition probabilities and expected rewards and determine the optimal policy by means of dynamic programming. We will further elaborate on this.

The state space is

$$\mathcal{S} = \{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)\}, \tag{8.9}$$

and the action space is

$$\mathcal{A} = \{(0,0),(0,1),(1,0),(1,1)\}. \tag{8.10}$$

We will present the policy and values as a table, where rows and columns correspond to the inventory level at LDC 1 and LDC 2 respectively.

For this example the optimal policy, obtained via dynamic programming is provided in Table 8.2a. As one can see the policy is symmetric, like we expected. The values per state corresponding to this policy are provided in Table 8.2b.

$\pi^*(I^1, I^2)$	0	1	2
0	(1,1)	(1,1)	(1,1)
1	(1,1)	(1,1)	(1,0)
2	(1,1)	(0,1)	(0,0)

(a) Optimal policy.

$V^*(I^1, I^2)$	0	1	2
0	-682.62	-678.66	-676.30
1	-678.66	-675.37	-673.66
2	-676.30	-673.66	-672.19

(b) Optimal values.

Table 8.2: Optimal policy and values for the model extension with parameters as in Table 8.1.

The Q-learning procedure resulted in the optimal policy as presented in Table 8.2a. To obtain this result we simulated 10.000 trajectories of length 1000. The learning is visualized in Figure 8.2. As one can see the simulated value from start state $(0,0)$ stabilized rather quickly. The optimal policy was found after approximately 3000 simulated trajectories. The final learned Q-value in state $(0,0)$ equals -682.54, which approximately equals the exact value in state $(0,0)$.

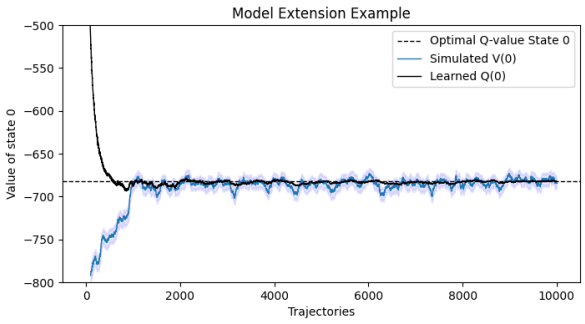


Figure 8.2: Visualization of the learning of Q-learning for the model extension case 3.

Case 4

In the last case we set all cost equal to 0, except for the emergency cost. As the only involved cost are the emergency cost, one wants to avoid that these cost have to be paid. In order to avoid unnecessary emergency cost, intuitively the LDC's should always be filled to the maximum capacities. Therefore the intuitive optimal policy is to always order the maximum amount of parts for both LDC's. The parameters are provided in Table 8.1.

This case did unfortunately not lead to the intuitive result. The resulting policy after 10.000 simulated trajectories is presented in Table 8.3a. In Chapter 7 we saw that for SKU 1 tabular Q-learning probably needs more time to converge for large state and action space. We therefore simulated 100.000 trajectories and monitored the progression of the policy. The policy after 100.000 simulated trajectories is presented in Table 8.3b. Figure 8.3 visualizes the progression of the learned Q-value in state 0, which seems to be fluctuating between -1400 and -1200.

It could be that the procedure needs even more trajectories to learn the intuitive policy, but it could also be that our intuition is misleading. Figure 8.3 does not provide the belief that more trajectories might solve the problem, but further research is needed to verify that.

$\pi^*(I^1, I^2)$	0	1	2	3
0	**(3,3)**	(3,2)	**(3,3)**	(3,2)
1	(2,2)	**(3,3)**	(2,3)	(2,1)
2	(1,3)	**(3,3)**	**(3,3)**	(3,2)
3	(2,3)	(2,2)	(2,3)	**(3,3)**

(a) Policy after 10.000 simulated trajectories.

$\pi^*(I^1, I^2)$	0	1	2	3
0	**(3,3)**	(3,2)	(2,3)	(3,2)
1	**(3,3)**	**(3,3)**	(2,3)	(2,1)
2	**(3,3)**	**(3,3)**	**(3,3)**	(3,2)
3	(2,3)	(2,2)	(2,3)	**(3,3)**

(b) Policy after 100.000 simulated trajectories.

Table 8.3: Policies for validation case 4 after 10.000 and 100.000 simulated trajectories.

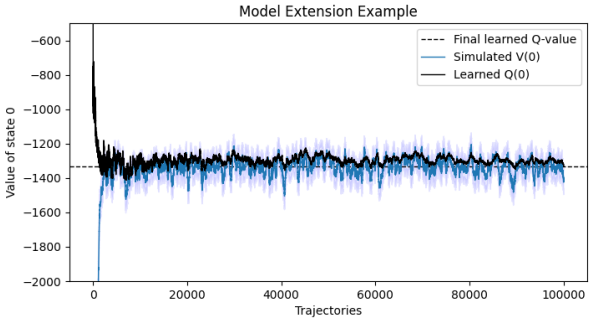


Figure 8.3: Visualization of the learning of Q-learning for the model extension, case 4.

8.3 Concluding Remarks

The proposed model extension adds complexity to the stylized model, and is therefore a step in the direction of a more realistic model. We validated the Q-learning method for the model extension using four cases. Only one of the cases, case 4, did not result in the intuitive policy. It might be that our intuition here is incorrect, or that the method simply needs more run-time. Further research is needed to confirm or disprove one or the other.

The other three cases did result in the intuitive or optimal policy. This indicates that the model extension could possibly work. Since we only touched upon the model extension, further research is needed for the model extension.

Chapter 9

Conclusions and Future Research

9.1 Findings

Recall that the main goal of this research was to gain a better understanding of RL, how it works and if it is applicable to spare parts inventory control. Spare parts inventory control problems can be modelled as MDPs, but that requires explicit knowledge of the model and parameters and can become intractable for large state and or action space. The benefit of RL over MDPs is that it can overcome both these problems.

We first considered a simplified single-item single-echelon (spare parts) inventory model, the stylized model. The optimal solution for the stylized model MDP was used a baseline for the RL methods discussed in this thesis. We discussed two RL methods and tested the applicability to the stylized model using a case provided by Philips Healthcare. In order to fit the application to the RL methods data analysis was performed to determine the demand distributions and cost parameters for three instances.

To test the applicability of RL in a more complex setting, where we do not have explicit knowledge of the model dynamics in terms of the transition probabilities and expected one step reward, we proposed a model extension.

9.1.1 Research and Performance RL Methods

The first RL method we have investigated is REINFORCE. This method directly produces a policy and therefore the results are easy to interpret. The main challenge we encountered when investigating REINFORCE was to gain a better understanding of the loss function and the elements that play an important role in the definition of the loss function.

Our investigation shows that the loss function is non-convex. Due to this non-convexity REINFORCE is likely to converge to a local optimum. For the Philips case REINFORCE indeed converges to (local) sub-optimal solutions, but the convergence was fast and there was only a small deviation of 1-2% from the optimal solution. REINFORCE thus performs quite well on the Philips case.

The second RL method we have investigated is Q-learning, a value-based RL method in which the values are updated iteratively via a specific update rule. The policy can be derived from the learned values. We investigated three versions of Q-learning. The first one, tabular Q-learning,

is the simplest form of Q-learing. The other two methods, double Q-learning and DQN, add complexity on two different levels.

Tabular Q-learning is guaranteed to converge under suitable conditions, but needs a lot of time to converge for large problems. We did not observe an advantage of double Q-learning over tabular Q-learning for the Philips case.

DQN seems to be the least consistent algorithm, as we did not observe a clear pattern in its behavior. However, DQN outperforms tabular and double Q-learning for a large instance of the Philips case. This could imply that DQN does not work for small instances, but that it might work for larger instances. Due to time limitations we did not investigate the DQN method as thoroughly as the REINFORCE, tabular and double Q-learning methods, therefore DQN is a suggested direction for future research.

9.1.2 Model Extension

As a first step to a more complex model we have proposed a model extension and applied tabular Q-learning to the extension. We validated the Q-learning method for the model extension using four intuitive cases. For three of the cases Q-learning resulted in the intuitive (or known optimal) solution. For the fourth case it is unclear whether it is a matter of run-time, intuition or if there is another problem. Since we only touched upon the model extension, we suggest further research in this direction.

9.1.3 Conclusion

All in all, we conclude the following. REINFORCE is the most consistent algorithm and finds a sub-optimal policy in a reasonable amount of time. Tabular Q-learning is the best algorithm for small instances, but needs too much time to converge for large instances. Double Q-learning does not have a significant advantage over Q-learning. DQN might be useful for large instances, as it outperforms tabular and double Q-learning for a large instance, but we cannot give a conclusive explanation for DQN's performance.

As a remark, we should mention that the performance of RL methods is very problem specific. RL methods do not come with performance guarantees in general, but for the single-item single-echelon spare parts inventory model REINFORCE and tabular Q-learning perform well.

9.2 Limitations and Future Research

In this section we discuss the limitations and suggestions for future research. In general we consider three topics: reinforcement learning methods, data analysis and model complexity.

9.2.1 Reinforcement Learning Methods

One of the limitations is that REINFORCE tends to converge to a local optimum. The results were close to optimality, but guarantees for performance could not be given. One direction for future research is to provide bounds on the performance of REINFORCE. The likely convergence to a local optimum is caused by the non-convexity of the loss function. Further research on the loss function might lead to new ideas on how to overcome this problem.

We were not able to give conclusive explanations to the behavior of DQN, as this method was less thoroughly investigated than the other methods discussed in this thesis. More investigation on this method is needed in order to further evaluate the performance.

This thesis provides little insight in the time that is needed for convergence. Note that this highly depends on the complexity of the problem. Some insightful visualizations are provided, but one could opt to develop rules of thumb for the time needed for convergence as further research.

In the DRL methods the neural network plays an important role. The structure of the neural network has impact on the performance. In this thesis the structure of the neural network is rather simple. Which structure is suitable, highly depends on the problem, so one could further investigate this to improve the performance. Tuning of the hyper parameters is a part of this.

In this thesis model-free RL methods were considered. Since we use a simulation model we have an implicit model of the environment and can control it. Therefore model-based methods can be used as well and might have a better performance. On the other hand, instead of using a simulation environment one could also use data or observations from an actual operating environment and feed that to the RL method.

Connecting the RL methods to the Philips case one could reconsider how to take into account the part availability target. Instead of tuning the parameters in order to reach the target, one could consider to include the availability target as a constraint.

9.2.2 Data Analysis

In the data analysis chapter the demand distributions for three items were fitted. However, we already remarked that it is challenging to fit distributions on data with limited support. With extra research the distribution fitting can be improved and justified. We did not apply a goodness-of fit test, as the regular goodness-of-fit tests are not directly applicable for discrete demand distributions. The conditions and assumptions to apply goodness-of-fit tests to discrete demand distributions need further investigation. Also, one could consider other demand distribution than the three distributions considered in this thesis to test the performance of the RL methods for other distributions.

9.2.3 Model Complexity

The last direction for future research is to add complexity to the models considered in this thesis to investigate if the RL methods could be useful in more complex situations. One can increase complexity in terms of the network structure, for example by adding more LDC's with interaction or lateral shipments, or by adding more echelons. Another way to increase complexity is in terms of the model parameters and the reward structure, for example by softening the stationarity assumption or differentiate between customer groups. A third option is to consider a mulit-item inventory model, where multiple items are correlated.

Bibliography

[1] Bansal, S., Calandra, R., Chua, K., Levine, S., and Tomlin, C. MBMF: Model-Based Priors for Model-Free Reinforcement Learning. *arXiv preprint arXiv:1709.03153* (2017). 7

[2] Bertsekas, D. P., and Tsitsiklis, J. N. Neuro-Dynamic Programming: An Overview. In *Proceedings of 1995 34th IEEE conference on decision and control* (1995), vol. 1, IEEE, pp. 560–564. 6

[3] Boyd, S., and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2004. 26

[4] Boylan, J. E., and Syntetos, A. A. Spare parts management: a review of forecasting research and extensions. *IMA Journal of Management Mathematics 21*, 3 (2010), 227–237. 10

[5] Cavalieri, S., Garetti, M., Macchi, M., and Pinto, R. A decision-making framework for managing maintenance spare parts. *Production Planning & Control 19*, 4 (2008), 379–396. 10

[6] Clark, A. J., and Scarf, H. Optimal Policies for a Multi-Echelon Inventory Problem. *Management Science 6*, 4 (1960), 475–490. 10

[7] Das, T. K., Gosavi, A., Mahadevan, S., and Marchalleck, N. Solving Semi-Markov Decision Problems Using Average Reward Reinforcement Learning. *Management Science 45*, 4 (1999), 560–574. 7

[8] Deisenroth, M. P., Neumann, G., and Peters, J. A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics 2*, 1-2 (2013), 1–142. 7

[9] Dmitrochenko, V. Allocation Decision-Making in Service Supply Chain with Deep Reinforcement Learning. Master's thesis, Eindhoven University of Technology, 2020. 10

[10] Gijsbrechts, J., Boute, R. N., Van Mieghem, J. A., and Zhang, D. Can Deep Reinforcement Learning Improve Inventory Management? Performance on Lost Sales, Dual Sourcing and Multi-Echelon Problems. *Performance on Dual Sourcing, Lost Sales and Multi-Echelon Problems (October 6, 2020)* (2020). 11, 12

[11] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT Press, 2016. 8, 9

[12] Hasselt, van, H., Guez, A., and Silver, D. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (2016), vol. 30. 30

[13] Huh, W. T., Janakiraman, G., Muckstadt, J. A., and Rusmevichientong, P. Asymptotic Optimality of Order-Up-To Policies in Lost Sales Inventory Systems. *Management Science 55*, 3 (2009), 404–420. 10

[14] Huh, W. T., Janakiraman, G., and Nagarajan, M. Capacitated Multiechelon Inventory Systems: Policies and Bounds. *Manufacturing & Service Operations Management 18*, 4 (2016), 570–584. 10

[15] Jogesh Babu, G., and Rao, C. R. Goodness-of-fit Tests When Parameters are Estimated. *Sankhyā: The Indian Journal of Statistics 66*, 1 (2004), 63–74. 44

[16] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research 4* (1996), 237–285. 6

[17] Kaynov, I. Deep Reinforcement Learning for Asymmetric One-Warehouse Multi-Retailer Inventory Management. Master's thesis, Eindhoven University of Technology, 2020. 10, 12

[18] Kemmer, L., Von Kleist, H., De Rochebouët, D., Tziortziotis, N., and Read, J. Reinforcement learning for supply chain optimization. In *European Workshop on Reinforcement Learning* (2018), vol. 14. 11

[19] Kingma, D. P., and Ba, J. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014). 9, 22, 36

[20] Kutanoglu, E., and Mahajan, M. An inventory sharing and allocation method for a multi-location service parts logistics network with time-based service levels. *European Journal of Operational Research 194*, 3 (2009), 728–742. 10

[21] Lemeshko, B. Y., Blinov, P. Y., and Lemeshko, S. B. Bias of Nonparametric Goodness-of-Fit Tests Relative to Certain Pairs of Competing Hypotheses. *Measurement Techniques 59*, 5 (2016), 468–475. 45

[22] Melman, G. J. Faster Service: Changing the Spare Part Supply Chain Network to Support Increasing Service Requirements. Master's thesis, Eindhoven University of Technology, 2021. 3

[23] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature 518* (2015), 529–533. 11, 30, 35, 36

[24] Nair, V., and Hinton, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML* (2010). 9

[25] Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. *arXiv preprint arXiv:1811.03378* (2018). 9

[26] Oroojlooyjadid, A., Nazari, M., Snyder, L. V., and Takác, M. A Deep Q-Network for the Beer Game: A Reinforcement Learning Algorithm to Solve Inventory Optimization Problems. *arXiv preprint arXiv:1708.05924* (2017). 5, 10, 11

[27] Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994. 13, 16

[28] Roberts, A. W., and Varberg, D. E. IV Convex Functions on a Normed Linear Space. vol. 57 of *Pure and Applied Mathematics*. Elsevier, 1973, pp. 88–120. 27

[29] Scarf, H. The Optimality of (S, s) Policies in the Dynamic Inventory Problem. In *Optimal pricing, inflation, and the cost of price adjustment*, E. Sheshinski and Y. Weiss, Eds. MIT Press, 1959. 10

[30] Schäl, M. Average Optimality in Dynamic Programming with General State Space. *Mathematics of Operations Research 18*, 1 (1993), 163–172. 17

[31] Sutton, R. S. Between MDPs and Semi-MDPs: Learning, Planning, and Representing Knowledge at Multiple Temporal Scales. *University of Massachusetts Amherst* (1998). 5

[32] Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 2018. 6, 7, 8, 17, 20, 26, 30, 31, 33

[33] Tsitsiklis, J. N. Asynchronous Stochastic Approximation and Q-Learning. *Machine Learning 16*, 3 (1994), 185–202. 31

[34] Tsitsiklis, J. N., and Van Roy, B. An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control 42*, 5 (1997), 674–690. 6, 35

[35] Tsitsiklis, J. N., and Van Roy, B. On Average Versus Discounted Reward Temporal-Difference Learning. *Machine Learning 49*, 2 (2002), 179–191. 6

[36] Van Den Bos, A. *Parameter estimation for scientists and engineers*. John Wiley & Sons, 2007. 27

[37] Van Hasselt, H. Double Q-learning. *Advances in Neural Information Processing Systems 23* (2010), 2613–2621. 30

[38] Van Roy, B. *Learning and Value Function Approximation in Complex Decision Processes*. PhD thesis, Massachusetts Institute of Technology, 1998. 6

[39] Van Roy, B., Bertsekas, D. P., Lee, Y., and Tsitsiklis, J. N. A Neuro-Dynamic Programming Approach to Retailer Inventory Management. In *Proceedings of the 36th IEEE Conference on Decision and Control* (1997), vol. 4, IEEE, pp. 4052–4057. 11

[40] Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989. 8, 31

[41] Watkins, C. J. C. H., and Dayan, P. Q-Learning. *Machine Learning 8*, 3-4 (1992), 279–292. 30, 31

[42] Whitehead, S. D., and Lin, L.-J. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence 73*, 1-2 (1995), 271–306. 7

[43] Williams, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning 8*, 3-4 (1992), 229–256. 8, 20, 22

[44] Xin, L. Understanding the Performance of Capped Base-Stock Policies in Lost-Sales Inventory Models. *Operations Research 69*, 1 (2021), 61–70. 10

[45] Zipkin, P. Old and New Methods for Lost-Sales Inventory Systems. *Operations Research 56*, 5 (2008), 1256–1263. 10

[46] Zipkin, P. On the Structure of Lost-Sales Inventory Models. *Operations Research 56*, 4 (2008), 937–944. 12

Appendix A

Bounded Rewards

Here we show that the expected rewards in the stylized model are bounded.

Note that we have $h, e \geq 0$, $0 \leq x \leq s_{max} < \infty$ for all $s \in \mathcal{S}$, and $0 \leq a \leq a_{max} < \infty$ for all $a \in \mathcal{A}$.

Since all rewards are negative we know

$$r(s,a) \leq 0. \tag{A.1}$$

Furthermore,

$$\begin{aligned}
r(s,a) &= -e\mathbb{E}[D] + e \cdot s - (e+h)s\mathbb{P}(D \leq s) + (e+h)\sum_{k=0}^{s} k\mathbb{P}(D=k) - O(a) && \text{(A.2)}\\
&\geq -e\mathbb{E}[D] + e \cdot s - (e+h)s - O(a) && \text{(A.3)}\\
&= -e\mathbb{E}[D] - h \cdot s - O(a) && \text{(A.4)}\\
&\geq -e\mathbb{E}[D] - h \cdot s_{max} - O(a_{max}) && \text{(A.5)}\\
&= -e\mathbb{E}[D] - h \cdot s_{max} - \tau_v \cdot a_{max} - \tau_f. && \text{(A.6)}
\end{aligned}$$

Thus,

$$|r(s,a)| \leq e\mathbb{E}[D] + h \cdot s_{max} + \tau_v \cdot a_{max} + \tau_f < \infty. \tag{A.7}$$

We can also bound the expected reward in the k^{th} step, $\mathbb{E}^{\pi}[R(S_{t+k}, a_{t+k}, D_{t+k}) \mid S_t = s]$. We use that $|r(s,a)| \leq C$, where $C := e\mathbb{E}[D] + h \cdot s_{max} + \tau_v \cdot a_{max} + \tau_f$.

$$\begin{aligned}
\mathbb{E}^{pi}[R(S_{t+k}, a_{t+k}, D_{t+k}) \mid S_t = s] &= \sum_{x \in \mathcal{S}} p_k^{\pi}(s,x)\mathbb{E}^{pi}[R(S_{t+k}, a_{t+k}, D_{t+k}) \mid S_t t + k = x] && \text{(A.8)}\\
&\leq \sum_{x \in \mathcal{S}} p_k^{\pi}(s,x)C && \text{(A.9)}\\
&= C, && \text{(A.10)}
\end{aligned}$$

where $p_k^{\pi}(s,x)$ is the probability to go from state s to state x in k steps under policy π.

Appendix B

Non-convexity of the REINFORCE Loss Function

For the small neural network in Section 4.2.3 the Hessian matrix of $L_0(s_t) = \log(\pi_\theta(0 \mid s_t))$ is

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{1}{2}s & -\frac{1}{2}s & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}s & -\frac{1}{2}s & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ \frac{1}{2}s & 0 & \frac{1}{2} & 0 & -\frac{1}{4}(s+1)^2 & \frac{1}{4}(s+1)^2 & -\frac{1}{4}(s+1)^2 & \frac{1}{4}(s+1)^2 & -\frac{1}{4}(s+1) & \frac{1}{4}(s+1) \\ -\frac{1}{2}s & 0 & -\frac{1}{2} & 0 & \frac{1}{4}(s+1)^2 & -\frac{1}{4}(s+1)^2 & \frac{1}{4}(s+1)^2 & -\frac{1}{4}(s+1)^2 & \frac{1}{4}(s+1) & -\frac{1}{4}(s+1) \\ 0 & \frac{1}{2}s & 0 & \frac{1}{2} & -\frac{1}{4}(s+1)^2 & \frac{1}{4}(s+1)^2 & -\frac{1}{4}(s+1)^2 & \frac{1}{4}(s+1)^2 & -\frac{1}{4}(s+1) & \frac{1}{4}(s+1) \\ 0 & -\frac{1}{2}s & 0 & -\frac{1}{2} & \frac{1}{4}(s+1)^2 & -\frac{1}{4}(s+1)^2 & \frac{1}{4}(s+1)^2 & -\frac{1}{4}(s+1)^2 & \frac{1}{4}(s+1) & -\frac{1}{4}(s+1) \\ 0 & 0 & 0 & 0 & -\frac{1}{4}(s+1) & \frac{1}{4}(s+1) & -\frac{1}{4}(s+1) & \frac{1}{4}(s+1) & -\frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \frac{1}{4}(s+1) & -\frac{1}{4}(s+1) & \frac{1}{4}(s+1) & -\frac{1}{4}(s+1) & \frac{1}{4} & -\frac{1}{4} \end{bmatrix}. \tag{B.1}$$

Since the value of $x^T Hx$ varies for different values of $s \in \mathcal{S}$ and vectors $x \in \mathbb{R}^{10}$ we generated 100 random vectors of length 10. For each vector we check whether the product $x^T Hx$ is less than or greater than 0 for several values of $s \in \mathcal{S}$. As one can see from the results in Table B.1 the hessian is neither positive nor negative semi-definite for $s \in \{0, 1, 2, 3, 4, 5\}$.

s	$x^T Hx < 0$	$x^T Hx > 0$	$x^T Hx = 0$
0	74	26	0
1	65	35	0
2	62	38	0
3	63	37	0
4	64	36	0
5	58	42	0

Table B.1: Results for $\boldsymbol{\theta} = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$, for 100 randomly generated binary vectors x of length 10.

We have also trained a neural network and used the resulting weights,

$$\boldsymbol{\theta} = (0, 2.1235, -0.8908, 0.0923, 1.0198, 1.5593, -1.3779, -1.7513, 0.6030, -0.9989, 0.2057), \quad \text{(B.2)}$$

as input for this study. Again for several values of s we checked whether there exists vectors for which $x^T Hx < 0$ and other vectors for which $x^T Hx > 0$. Here we see in Table B.2 that for $s = 0$ and $s = 1$ we have vectors for which $x^T Hx < 0$ and vectors for which $x^T Hx > 0$. This proves that the Hessian is neither positive nor negative semi-definite.

s	$x^T Hx < 0$	$x^T Hx > 0$	$x^T Hx = 0$
0	65	35	0
1	83	17	0
2	100	0	0
3	100	0	0
4	100	0	0
5	100	0	0

Table B.2: Results for $\boldsymbol{\theta}$ as defined in Equation (B.2), for 100 randomly generated binary vectors x of length 10.

Appendix C

Dynamic Programming

Here the pseudo-code for the dynamic programming algorithms as discussed in Section 3.2 is presented.

C.1 Policy Iteration

The policy iteration algorithm presented in Algorithm 7 uses the PolicyEvaluation and PolicyImprovement functions from Algorithm 8 and 9.

Algorithm 7 Policy Iteration

Initialize parameters γ, θ
Initialize $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}$ for all $s \in \mathcal{S}$ arbitrarily
policyStable = False
while policyStable = False **do**
    V = PolicyEvaluation(env, V, π, γ, θ) ▷ See Algorithm 8
    policyStable, π = PolicyImprovement(env, V, π, γ) ▷ See Algorithm 9

Algorithm 8 Policy Evaluation

function PolicyEvaluation(env, V, π, γ, θ)
    converged = False
    while converged = False **do**
        $\Delta = 0$
        for all $s \in \mathcal{S}$ **do**
            $oldV(s) = V(s)$
            $V(s) = r(s, \pi(s)) + \gamma \sum_{j \in \mathcal{S}} p^{\pi(s)}(s, j) V(j)$
            $\Delta = \max(\Delta, |oldV(s) - V(s)|)$
            if $\Delta < \theta$ **then** converged = True
    return V

Algorithm 9 Policy Improvement

function PolicyImprovement(env, V, π, γ)
    policyStable = True
    for all $s \in \mathcal{S}$ **do**
        $oldA(s) = \pi(s)$
        $\pi(s) = \arg\max_{a \in \mathcal{A}} \{r(s,a) + \gamma \sum_{j \in \mathcal{S}} p^a(s,j) V(j)\}$
        if $oldA(s) \neq \pi(s)$ **then** policyStable = False
    return policyStable, π

C.2 Value Iteration

Algorithm 10 Value Iteration

Initialize $V(s) \in \mathbb{R}$ for all $s \in \mathcal{S}$ arbitrarily
procedure ValueIteration(env, V, π, γ, θ)
    while $\Delta \geq \theta$ **do**
        $\Delta = 0$
        for all $s \in \mathcal{S}$ **do**
            $oldV(s) = V(s)$
            $V(s) = \max_{a \in \mathcal{A}} \{r(s,a) + \gamma \sum_{j \in \mathcal{S}} p^a(s,j) V(j)\}$
            $\Delta = \max(\Delta, |oldV(s) - V(s)|)$
Output a deterministic policy such that $\pi(s) = \arg\max_{a \in \mathcal{A}} \{r(s,a) + \gamma \sum_{j \in \mathcal{S}} p^a(s,j) V(j)\}$

Appendix D

Regression Transportation Cost

To determine the fixed and variable cost for the chosen SKU's in Chapter 6 a regression analysis was performed. Table D.1 shows the total cost to send 1 to 10 items at once per SKU. These values are derived from the UPS rate structure data set. The cost is the dependent variable and the weight is the independent variable. The results of the regression analysis are provided in Table D.2.

	SKU 1		SKU 2		SKU 3	
Quantity	Weight	Cost	Weight	Cost	Weight	Cost
1	0.2	3.61	4.1	7.73	117	42.12
2	0.4	3.61	8.2	9.47	234	84.24
3	0.6	4.11	12.3	10.9	351	126.36
4	0.8	4.11	16.4	12.14	468	168.48
5	1.0	4.11	20.5	13.02	585	210.60
6	1.2	5.11	24.6	13.49	702	252.72
7	1.4	5.11	28.7	13.96	819	294.84
8	1.6	5.77	32.8	14.65	936	336.96
9	1.8	5.77	36.9	15.98	1053	379.08
10	2.0	5.77	41.0	16.93	1170	421.20

Table D.1: Total weight and cost to send items with the UPS service.

	SKU 1	SKU 2	SKU 3
R Square	0.925	0.968	1.0
Intercept (Fixed cost τ_f)	3.146	7.725	0.0
Quantity (Variable cost τ_v)	0.284	0.928	42.12

Table D.2: Output regression analysis to determine fixed and variable transportation cost for the UPS service.

Appendix E

Results Stylized Model Extra Figures SKU 1

In Chapter 7 we provided Figure 7.2b that shows that tabular and double Q-learning have not converged after 10.000 simulated trajectories. Here, in Figure E.1 we see that both methods also have not converged after 100.000 trajectories. As the learned Q-values are decreasing we suspect that the process needs more time for convergence.

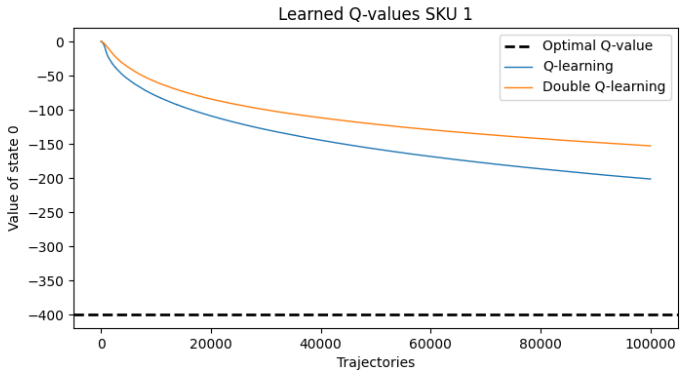


Figure E.1: Learned Q-values for 100.000 simulated trajectories for SKU 1.

In Figure E.2 the learned policies for each method for the entire state space are presented.

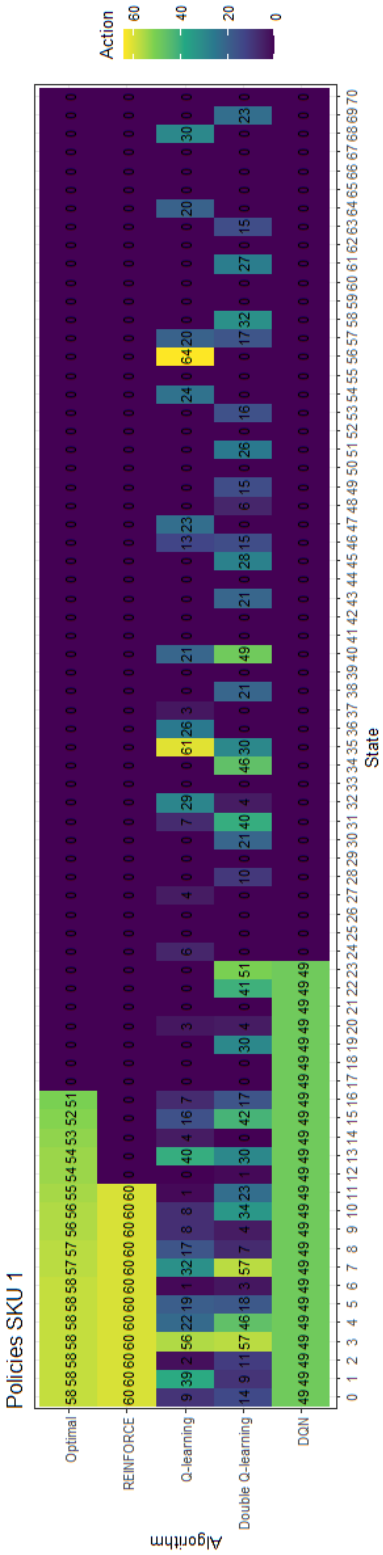


Figure E.2: Learned policies for SKU 1.