Reinforcement Learning-Based Ordering

A study at Syncron AB

Master Thesis

AUTHORS

Samuel Dahn, sa2606da@student.lu.se
Theodor Hallbeck, th8478ha@student.lu.se

SUPERVISOR

Johan Marklund

THESIS PERIOD

Jan 2026 – Jun 2026

Abstract

This thesis investigates the applicability and performance of reinforcement learning (RL) for inventory management of spare parts in a single-echelon setting. Traditional inventory control methods, such as (R,Q)-policies, typically decouple forecasting and ordering decisions, which may lead to suboptimal system-wide performance, particularly in environments characterized by intermittent and stochastic demand.

To address these limitations, a reinforcement learning-based ordering model is developed and evaluated using both simulated and real-world data provided by Syncron AB. The proposed model is a Dueling Double Deep Q-Network (DDDQN) architecture incorporating enhancements such as prioritized experience replay, experience buffer modification, and early stopping to improve training efficiency and stability.

The performance of the RL-model is benchmarked against a standard periodic review (R,Q)-policy using backorder and holding cost. Results indicate that while RL demonstrates potential in capturing complex system dynamics and achieving competitive performance in certain scenarios, its effectiveness is highly dependent on data availability, demand characteristics, and computational constraints. In particular, challenges related to intermittent demand, policy convergence, and generalization to unseen environments restrict consistent outperformance of traditional methods.

The findings suggest that RL based approaches can complement existing inventory control methods but cannot yet fully replace them in practical applications. This highlights the importance of hybrid decision frameworks that leverage the strengths of both data-driven and analytical methods. The thesis concludes with recommendations for future development, including model scalability, improved data utilization, and more robust evaluation across varying demand patterns.

Preface

This master thesis was part of the end of five years of studies in industrial engineering and management at Lund University, Faculty of Engineering, LTH, for the authors. The work was carried out in close collaboration with Syncron during the spring of 2026.

We sincerely thank our supervisor at Lund University, professor Johan Marklund, for interesting discussions, constant support, and constructive feedback. This has been invaluable in moving the thesis forward.

In addition, we express our sincere gratitude to our supervisors at Syncron, Staffan Theander, Lina Johansson, Paweł Samoraj, and Edyta Mazurek. It has been a pleasure working with you, and we thank you for your insights, active involvement in our thesis, and for enabling our model development.

Samuel Dahn & Theodor Hallbeck, Lund 2026

Contents

1 Introduction

This master thesis aims to evaluate and apply Reinforcement Learning (RL) in the inventory management of spare parts at Syncron AB. To outline the challenges associated with this implementation, this introduction provides background on existing optimization methods, relevant machine learning use cases, and the work conducted by Syncron so far.

1.1 Background

1.1.1 Current optimization methods

With current methods for handling inventory control systems, optimization is often split into forecasting and ordering systems and is addressed individually (Goltsos et al. 2022, Andersson et al. 2010, Axsäter 2015). Consequently, this section will decouple them as well.

Forecasting When ordering, there is almost always a lead-time between the time of ordering and the time of delivery (Axsäter 2015). Demand forecasting tries to manage this lead-time by predicting future demand. Furthermore, since demand is usually stochastic, there is a need to account for uncertainty in the forecast. If uncertainty increases, this will also increase the safety stock needed to manage the risk of stock-out. Forecasting is often done by probabilistic modeling of demand based on historical demand data. There are multiple models for forecasting depending on the patterns (e.g., trends and seasonality) detected in demand data. In today's market with more complex demand, there are multiple internal (e.g., item correlation) and external factors that impact demand in ways not captured by the ruling class of forecasting models (Abolghasemi et al. 2020).

Ordering Ordering systems captures the main decision-making in inventory control. They are commonly expressed through inventory control policies that regulate ordering decisions. These decisions are commonly based on the inventory position (IP = stock on hand + outstanding orders - backorders) (Axsäter 2015). Two commonly used control policies in the literature and in practice are (R,Q)- and (s,S)-policies. For an (R,Q)-policy, R is the reorder point and Q is the order quantity. When $IP \leq R$, an order of quantity Q is placed. For the (s,S)-policy, s denotes the reorder point, and S is the order up to level. When $IP \leq s$, an order is placed that fills up the IP to S. For both (R,Q)- and (s,S)-policies, there are sub-policies that are classified as one of the two main policies but handle slightly different ordering systems (Axsäter 2015). The choice of policy parameters (e.g., R,Q,s,S) depend on demand and lead-time characteristics, as well as cost and service targets, and are often obtained via analytical approximations or numerical optimization.

The frequency of monitoring the inventory is an additional factor which ordering decisions depend on. If monitored continuously, this is defined as *continuous review* and, if monitored periodically, *periodic review* (Axsäter 2015). The review period T, the time between two reviews, for periodic review affects the choice of approach for inventory management. For example, for T close to 0, the inventory system approximates to continuous review.

Inventory control problems are often cost minimization problems with service level constraints (Axsäter 2015). Parameter choices are thus evaluated on the basis of incurred cost. Costs usually included when optimizing the control are holding costs, setup or ordering costs, and shortage costs. Most inventory control methods assume complete backordering. If, on the other hand, lost sales are assumed to occur, lost sales cost has to be included as well. Real world factors, like lost sales or quantity dependent ordering costs, are difficult to portray probabilistically, and thus, current models can in some cases be found insufficient.

1.1.2 Spare parts, does it differ?

The mentioned strategies for inventory optimization perform best when demand data can be accurately modeled. For spare part modeling, with often low and lumpy demand, methods directed towards more frequent demand often underperform (Pinçe et al. 2021, Lengu et al. 2014). Additionally, the time critical nature and complex inventory management systems, for example with batch ordering, further add to the complexity. As noted by Andrey Meshalkin, Syncron (2026), it is essential to "get the right part to the right place at the right time". As presented in Table 1, there are several differences between supply chains for spare parts and manufacturing.

Table 1: Differences between manufacturing and aftermarket supply chains (Andrey Meshalkin, Syncron 2026)

Parameter	**Manufacturing Supply Chain**	**Aftermarket Supply Chain**
Nature of demand	Stable, planned demand driven by a defined sales and production plan	Intermittent, unpredictable demand driven by failures and service needs
Demand drivers	Finished goods forecasts & customer orders	Breakdowns, service events, SLAs, maintenance patterns
"Required by" date	Clear production date from assembly sequencing	ASAP to meet service needs
Number of SKUs	Limited to active BOM components for current products	Highly diversified, slow-moving parts portfolios
Physical scope	Production sites and a small number of central distribution points	Global OEM DCs, distributors & thousands of dealer sites
Inventory strategy	Just-in-Time: minimize inventory and maximize material flow	Just-in-Case: preposition inventory close to consumption
Reverse logistics	Minimal or not relevant	Significant: returns, repairs, warranty flows
Lifecycle	Ends at end-of-production	Extends to end-of-service (often decades later)

1.1.3 Possibilities and Limitations with Machine Learning

Possibilities and Advantages with RL The interest, usage, and publications of machine learning (ML) have been rapidly increasing within Supply Chain Management (SCM) (Babai et al. 2025). Advantages with ML, such as processing large amounts of data in a relatively short time, improve several processes, for example, demand forecasting and inventory management.

Especially demand forecasting has been relevant for the application of ML where advantages such as mitigating the bullwhip effect in multi-echelon supply chains have been seen (Babai et al. 2025). The improvement in using ML methods in comparison to conventional methods can be largely ascribed to the ability to process more and different types of data than before. For example, using both exogenous (such as price fluctuation and customer behavior) and endogenous (historic demand) data rather than just endogenous to predict future demand. This enables pattern recognition and possibly more precise predictions.

More generally, ML gives the largest performance improvements when additional information, often exogenous, can be considered and used for predictions or decisions. This is due to ML models ability to find connections that are not possible or difficult for a human to find. Additionally, these models can be helpful when the sheer size of the data is too large to process manually or with already existing methods. However, this advantage is likely limited for inventory management of spare parts. Demand of spare parts is mainly driven by failure rates and the number of machines using a part (Axsäter 2015), a generally less complex demand situation than other parts. For that reason, the advantages of exogenous data in ML for spare parts inventory management is likely less than for other parts; nonetheless, it should not be disregarded.

Limitations with RL In order to obtain an ML model that improves performance in the mentioned settings, new and future processes and data must be somewhat similar to historical processes and training data (often historic data, for example incoming orders or customer behaviors) (Ohlsson et al. 2025).

The performance of RL models for general unseen data depends on whether new input data come from the same or similar processes (Li et al. 2025). Therefore, rapidly changing processes pose a problem for ML-models. For these scenarios, ordinary methods to increase generalization does not significantly improve the performance. Of course, this depends on the closeness of training data/process and new testing data. Additionally, insufficient training data similarly affect the generalization performance negatively. This follows from the model being unable to draw conclusions and find relevant patterns from the data. These two aspects can interact with each other since a rapidly changing process results in historic data quickly becomes irrelevant and unsuitable for model training, thereby limiting the usable data.

1.1.4 Machine Learning Paradigms

Within machine learning, there are three main learning paradigms that differ in several aspects, such as what type of data they train on (Wang 2025). The three paradigms are supervised learning, unsupervised learning and reinforcement learning. While these three are regarded as main paradigms, semi-supervised and self-supervised can be regarded as either additional paradigms or sub-methods to the main three.

While these paradigms rely on different mathematical algorithms and approaches, hybrid solutions where different paradigms are used for different stages in training and/or data processing can be beneficial (Wang 2025).

Supervised Learning
Supervised learning is focused on labeled training data, where a correct output always can be defined (Wang 2025). Suitable problems are often classification problems, regression problems, or binary decision making where the model aims

at recognizing patterns in training data to later apply the knowledge to new data and make estimations of the classification or regression.

As mentioned, this method assumes the data have been labeled beforehand which is often done by a human who is assumed to 'supervise' the learning (Wang 2025). This supervision results in a relatively efficient training procedure in comparison to other paradigms, while also implying limitations for the application. The optimization and learning for these models use a loss-function which compares the model output with the correct/expected output and explores what changes to the model give the best improvement for model output.

Unsupervised Learning
In contrast to supervised learning, unsupervised learning does not require labeled data and a definition of what output is deemed correct (Wang 2025). This paradigm is suitable for problems such as clustering or association problems. For these tasks, labeling training data might be difficult or impossible for various reasons such as a vast amount of data, patterns not apparent to human observers, etc. This results in unsupervised learning not using a loss function but instead focuses on finding patterns, dependencies, and correlations in data.

Reinforcement Learning (RL)
Different from supervised and unsupervised learning, reinforcement learning fits problems where a series of actions gives a result (Wang 2025). The result of the actions gives a reward for the ML-model (often called agent in RL) that can be positive, negative or neutral. Therefore, the RL-agent does not minimize the loss but rather aims to maximize the reward.

The framework for such models is based on components such as state space, action space, reward signal and policy (Wang 2025). The state space defines all possible states and the relevant information that the agent needs to make a decision. The action space denotes all legal moves for a model for a given state. The policy can be compared to a "thought process" and this is what gives an action for the given state. Lastly, the reward signal is what gives feedback to the agent and rewards or punishes previous actions.

1.2 Problematization

1.2.1 Disadvantages with current optimization methods

The sectioning of forecasting and ordering systems, simplifies a complex problem. This poses the risk of finding suboptimal solutions for the whole problem. For that reason, it could be relevant to reevaluate current optimization methods and explore alternative paths that handle inventory management as a whole.

1.2.2 Applicable advantages with ML in inventory control

As mentioned in section 1.1.3, one of the main advantages with machine learning is the possibility to consider a broader amount of data, options, decisions and constraints. This is done by disregarding traditional handling of inventory management and instead being strictly data driven allowing complex state dependent policies and actions. For spare parts and inventory management, this could include data such as fluctuating price, customer behavior on multiple supply options, competitor behavior or supply chain disruption. Even dependencies between different parts could be considered to optimize ordering. These dependencies could for example be parts for the same machine with similar failing patterns or supply chain advantages such as reduced transport cost when ordered together. By considering a wide range of variables, it is possible to find ordering patterns that are better than those for standard (R,Q)-policies.

The problems with ML training still pose the risk of limiting the possible advantages. These problems are mainly due to the lack of historical and reliable data as well as changing demand patterns. For spare parts where the demand is generally low and lumpy, these problems are obvious. The exact extent of these problems has yet to be explored, but they pose the risk of either limiting or erasing the possible improvement for ordering procedures, specifically for spare parts.

1.2.3 Why RL? - Evaluation of ML paradigms

The differences of the ML paradigms is, as previously mentioned, mainly focused on the type of input data. For inventory management, reinforcement learning is often considered a suitable option (Gijsbrechts et al. 2022). With RL, the model allows for a delay in reward, which is applicable for example for stock outs or similar events that are results of many past decisions. An RL model is designed to maximize the reward rather than finding a correct output, and this reward can, for example, be connected to costs in a warehouse or supply chain to train the model to reduce costs.

1.2.4 Continuing the work by Syncron

The RL model constructed by Syncron today lacks many of the complexities sought after when implementing an RL model. The simulated situation could therefore be compared to a toy-problem focused on showcasing potential rather than applicability. Although this is the case, the built model serves as a stepping stone for further research and experiments in the area. The model should be extended for applicability upon real life data, with stochastic demand and multiple items. To evaluate performance, it should be compared with an industry used method of optimization, an (R,Q)-policy.

The difference in complexity of inventory management problems regarding single versus multi echelon is vast. By considering several installations simultaneously,

close to optimal solutions can often be found, but with increased complexity. To ensure feasibility in the RL environment and model, limitation to complexity has to be considered. For that reason, it is relevant to, in this thesis, limit the RL-model to single echelon inventory management.

By scaling the model, Syncron's knowledge regarding the applicability of RL on inventory optimization and RL's performance in comparison to the leading ordering policies of today will deepen. Thus, this work will be able to deliver a next step recommendation for Syncron regarding the implementation of RL for inventory optimization.

1.3 Purpose

The purpose of this thesis is to evaluate the potential and obstacles of using an RL based model for inventory management of spare parts. Additionally, it will be investigated how an RL based model can be implemented in regards to what information could be handled and how decision making could be eased.

1.3.1 Research questions

To fulfill this purpose, the following research questions are to be investigated:

RQ1: May a reinforcement learning based model consistently outperform (R,Q)-policies for inventory management of single-echelon intermittent spare parts demand?

RQ2: How could Syncron continue the work with reinforcement learning-based ordering?

1.3.2 Limitations

- The thesis will only focus on the paradigm reinforcement learning within machine learning.
- The thesis will only consider inventory management of spare parts with intermittent demand.

2 Methodology

2.1 Research Strategy

The choice of thesis methodology is highly dependent on the characteristics of the planned research (Höst et al. 2006). This thesis will not only explore the problem of inefficient inventory management, but also aim to deepen the understanding of possibilities and obstacles by using reinforcement learning in inventory management. Consequently, the methodology will have two parts with an exploratory approach followed by a problem solving approach. Since the thesis has two differing aims and approaches, it will include two sub-studies with different methodologies, literature review, and operations research.

The major part of this thesis will focus on the use of RL in inventory management of spare parts. The RL work will follow two directions. First, an RL-based policy will be compared with existing methods to assess its performance and identify key strengths and limitations. Second, different RL design choices will be investigated to determine how the RL-policy can be improved while managing computational requirements. The latter part will have a flexible approach in which the exact procedure is not predetermined but allows for an iterative process and the methodology is continuously updated (Höst et al. 2006).

2.2 Literature Review

To complement the practical model implementation and explore the research landscape, a literature review will be done. The systematic approach presented in Höst et al. (2006) will be adopted to handle the published research. A literature review is an iterative process in which the determination of key phrases, search, selection, evaluation, and compilation are alternated. To ensure a comprehensive coverage of the topics of the literature review, the search process will systematically include a broad search (i), a selection (ii), and a deep search (iii).

(i) - Broad Search
In the initial stages of the literature review, a broad approach will be used in accordance with Höst et al. (2006). This aims to capture a wide range of performed studies by using a broad range of key phrases and searches in Lund University's database Finn. Finn searches encompass 500 different databases, of which Inspec, Scopus, and IEEE gives the most number of results. In addition, discussions with university and company supervisors about field experience resulted in searches for experienced researchers within relevant fields.

(ii) - Selection
Following the broad search, a selection of relevant papers, articles, and other sources were chosen for deeper study. This selection was the basis for an even further and deeper search for sources as per Höst et al. (2006).

(iii) - Search Deep
Based on the selected sources obtained during the broad search, a deep search was performed to obtain the most relevant sources. This was done in the Finn database by Lund University. The deep search was confined by developing more thorough combinations of key phrases while also using reference lists in relevant articles and authors within the subjects of interest.

During the literature review, the following keywords were used: *reinforcement learning, inventory management, inventory control, ordering, forecasting, spare parts.*

2.3 Operations Research

The other leg of the thesis is the development and evaluation of an RL-model. One way to do this systematically is using *Operations Research* (OR) methodology (Hillier 2005). The OR methodology as described by Hillier (2005) follows a 9 step process in a full fledged OR-study. This thesis will not include all nine steps, but will adopt a simplified version that accommodates the scope of the study. The full 9 step process:

1. **Define the problem of interest**
   In operations research, most practical problems encountered are loosely defined and often described in an imprecise manner (Hillier 2005). Thus, the need is firstly to study the system in question and define the confinements of the problem and the objectives when proposing a solution.

2. **Gather and organize relevant data**
   The quality of the data gathering and organization process affects the ability of an OR study to reach its targets. For a good OR study, the collected data should be relevant. The data will then guide the model creation and determine how representative of the problem the model will become. Thus, the data needs to be extensive enough to encompass the full problem definition.

3. **Use descriptive analytics to analyze big data**
   Descriptive analytics encompasses the use of data formulation to accurately analyze the data and form conclusions. The goal of descriptive analytics is to better understand both past and real time information and can be used as a stepping stone to a succeeding application of predictive or prescriptive analytics.

4. **Use predictive analytics to analyze big data**
   While descriptive analytics focuses on past and current information flow, predictive analytics, as the name implies, puts the spotlight on future events, trends, and behaviors. In the age of big data, the access to large

data sets provides the opportunity for data driven statistical forecasts using, for example, different types of regressions and exponential smoothening. Furthermore, ML is a powerful tool that can be used for applications such as predictive analytics and forecasting. Machine learning is especially efficient for analyzing big data, as it relies on large data sets to learn and exploit patterns.

5. **Formulate a model to begin applying prescriptive analytics**
   Prescriptive analytics is the process of using prescriptive models, for example, an optimization model, to aid decision making. For a satisfactory model formulation, there has to be a high correlation between model prediction and real world outcome. Thus, the models balance tractability with precision to provide a computationally efficient and applicable solution.

6. **Learn how to derive solutions from the model**
   After model formulation, the next step is to find a procedure for deriving solutions from the model. In the OR field there are many standard algorithms to solve well formulated OR problems. When learning how to derive solutions from the model, it is important to keep in mind the objective of the OR process, which is often not to find the optimal solution but a solution that is satisfactory. Focusing on good solutions rather than optimal balances the cost of the study with the results to maximize the net benefits of the study.

7. **Test the model**
   When the model has been formulated and solutions can be derived from it, the model has to be tested. Like all programs, the model will inevitably have some bugs that need to be corrected. This is done by testing the model and iteratively removing detected bugs. Furthermore, the validity of the model has to be questioned. *Model validation* largely encompasses ensuring that the model exhausts the problem definition and returns a result in line with what is expected. Another systematic way to test the model is using a *retrospective test*, which involves using historical data to determine how well the model would have performed if it had been used. While this provides insight into actual performance of the model, the validity of a retrospective test deteriorates if the historical data has been used when formulating the model. Moreover, the historical data might not be representative of the future upon which the model should be applied.

8. **Prepare to apply the model**
   Before implementation, if the model is to be used repeatably, a well-documented system for model application should be developed. This ensures that the model can be used by others and that a systematic approach is applied each time the model is used.

9. **Implementation**
   The actual benefits of an OR study are gathered first in the implementation phase. It is therefore essential for the team responsible to oversee the process of implementation to ensure solutions are accurately translated into operating procedures. Furthermore, implementation of an OR model should be made in cooperation with management. Since the operational reality hinges on management, their support and perceived ownership over the study greatly impacts how successful the model will be deemed.

The model developed in this thesis has OR-characteristics. In contrast to the models described in Hillier (2005), the developed model does not apply a mathematical framework but rather an RL algorithm and thus, not all steps of the 9 step process are relevant. Consequently, consistent with the purpose (section 1.3), this thesis will focus on steps 1, 2 and 4 through 7. Their application in a RL development setting are clarified in 6 steps.

1. **Define the problem of interest**
   The problem is defined through the problematization and purpose in section 1.2 and 1.3.

2. **Gather and organize relevant data**
   As described, the model will first use deterministic and stochastic data generated for each iteration. The subsequent development of the model will use historical data. Syncron AB will provide anonymized sales data that in turn will be used to subject the model to real life data. Data relevance will be ensured through cooperation with Syncron.

3. **Use predictive analytics to analyze big data**
   Since the model is based on RL, a subarea of ML, predictive analytics is a part of the model. It will use the input to estimate the value of different states and actions and, from this estimation, choose the best action.

4. **Formulate a model to begin to apply prescriptive analytics**
   For this thesis, formulating the model means applying RL-theory to practice and writing the code for the model. This is done using ML code libraries in Python, namely PyTorch and the Open AI gymnasium environment.

5. **Learn how to derive solutions from the model**
   For RL, this is a major part of model development. The results of the model are dependent on how well trained it is and how much time it has had to learn. Furthermore, issues like overtraining and undertraining complicate how quickly the model converges toward a plausible solution space. In general, neural networks have low traceability in decision making and relations between hidden nodes are hard to follow. It is therefore important to validate solutions in a structured way.

6. **Test the model**
   Finally, the model will be tested. This will be done using test data. The

test data is derived from the data set which is split into training data and test data. During and especially after model training, the model will be tested using test data and give an insight into performance of the model and if it is overtrained.

2.4 Research Quality

The quality of research is central to the validity of the thesis and can be evaluated using different criteria. Höst et al. (2006) states that there are different categorizations available and Yin (2018) presents four quality criteria for case studies. The same four aspects of validity are presented by Runeson & Höst (2009). Thus, the four aspects will be adopted as a quality tool for this thesis. These are *Construct Validity, Internal Validity, External Validity,* and *Reliability.*

- **Construct Validity** evaluates whether the study accurately represents what the researcher is trying to accomplish. If the measures used are not sufficiently operational or introduce subjectivity, this threatens the validity of the research.
- **Internal Validity** assesses whether the investigated factors are correlated with factors not investigated. If factor correlation between two factors is investigated, the risk of internal validity is that a third factor might be impacting the relation.
- **External Validity** deals with the generalization of the research. It evaluates whether the findings are relevant in other settings and cases.
- **Reliability** aims to ensure findings independent from the researchers performing the study. If the research is repeatable by external researchers, the results should be the same.

Several countermeasures against threats to validity have been used to obtain a well grounded study result.

Firstly, a wide range of sources were used to ensure an appropriate interpretation of the information gathered. Furthermore, the applicability of the research is tested using a RL-model. To ensure results with high construct validity, a representative data set is obtained from Syncron to ensure that the study is evaluated on the correct data basis.

Secondly, to avoid low internal validity, the model is tested with multiple data samples. Thus, data triangulation is used to discover possible uninvestigated factors that are affecting the results. Thirdly, to ensure that the study generalizes well, external validity is addressed. This is performed by testing the model after model training using a test data set. Thus, possible overtraining is visualized. Moreover, the literature study is broadened to include neighboring

research areas. Consequently, the composite of the research evaluated is generalized over relevant areas of research.

Finally, reliability of the study is obtained by continuous documentation of the research process and transparency regarding sources used and data sets and algorithms used for the model.

3 Theory

3.1 Inventory management

3.1.1 Ordering policies

(R,Q)-policy A widely utilized inventory control strategy is the (R,Q)-policy. A challenge in the implementation is the determination of the policy parameters. In practice, it is common to calculate the order quantity (Q) before establishing the reorder point (R). This may be achieved by using the Economic Order Quantity model (EOQ), as shown in Equation 1, where A represents the ordering cost, d is the demand per time unit, and h is the holding cost per unit and time unit (Axsäter 2015). The EOQ model relies on specific assumptions and requires all input parameters to be known or accurately estimated, and therefore industry often uses, heuristic methods for real-world application (Johansson 2026). Determining the reorder point is commonly done by using a desired service level for a product (Axsäter 2015). The method can be either an analytical approach or based on simulations. Both of these methods are expected to give similar results if the simulation model and the analytical model are based on the same assumptions and include the same information.

$$Q^* = \sqrt{\frac{2Ad}{h}} \tag{1}$$

(s,S)-policy For continuous review and continuous demand situations, an (s,S)-policy is equivalent to an (R,Q)-policy. However, this is not always the case for real world applications. The difference shows when the inventory position drops below the reorder point (s) and therefore the order quantity differs. Additionally, an (s,S)-policy can be proven optimal, in contrast to an (R,Q)-policy. This is a theoretical advantage for the (s,S)-policy, but the practical cost difference is often limited (Axsäter 2015).

External Constraints One factor that affects both the (R,Q)- and (s,S)-policies is constraints from suppliers (Johansson 2026). These constraints may be order quantity, where orders have to be placed in a minimum quantity, multiples of a certain quantity, and constraints on when placed orders are shipped. If orders are shipped from suppliers once per day, a continuous review system will perform similarly to a daily review system. In addition, receiving deliveries might also be the subject of constraints. For example that deliveries may not be handled during weekends.

3.1.2 Service Levels

A common approach in inventory management to determine suitable parameters in ordering policies is to use service levels (Axsäter 2015). There are three

main types of service levels with slightly different focus, but all with the aim of quantifying the service to the customer.

S1 - Probability of no stockout per order cycle
S1 is the simplest of the three service levels to estimate. It can be interpreted as the probability that an order will arrive before stockout. A disadvantage of S1 is that the batch size is not considered, and therefore the amount and frequency of unmet demand may vary for the same S1 (Axsäter 2015).

Nonetheless, S1 is relatively widely used in industry due to its simplicity, although it is often not recommended (Johansson 2026). For an (R,Q)-policy with a given order quantity, the reorder point for S1 can be calculated by incrementally increasing the reorder points from $-Q$ until a desired service level is achieved (Axsäter 2015).

S2 - "fill rate" - Fraction of demand satisfied immediately from stock on hand
S2 is a widely used service level but more complex to calculate analytically than S1 (Axsäter 2015). The exact calculations vary depending on the distribution of the demand and the replenishment lead time.

S3 - "ready rate" - Fraction of time with positive stock on hand
The ready rate, or S3, is in cases with continuous or Poisson demand the same as S2 (Axsäter 2015). The two service levels differ when the stock on hand is positive but insufficient to satisfy customer demand. This, for example, occurs when customers place large orders. A low inventory in combination with large customer orders may therefore result in a high ready rate and a lower fill rate.

3.1.3 Backorder and holding cost

An alternative to using service levels is to balance backorder costs (b) and holding costs (h) (Axsäter 2015). In this approach, costs are estimated and optimized to determine a reorder point that minimizes the total expected cost. Holding costs typically represent the cost of tied-up capital and possible additional costs such as handling and insurance. However, backorder costs are generally more difficult to estimate, which often limits their practical application compared to service levels. Backorder costs can be applied either per unit, or per unit and time unit. For spare parts, the latter is generally more appropriate (Axsäter 2015). Nevertheless, for S_2 it is possible to find a service level that yields the same reorder point as a specific backorder cost. Given that the cost function is convex with respect to the reorder point, this relationship is presented in Equation 2. For continuous demand, the left inequality becomes a strict equality.

$$S_2(R) \leq \frac{b_1}{h + b_1} < S_2(R + 1) \tag{2}$$

3.1.4 Difference between demand and sales data

One concern that arises when forecasting future demand is the difference between demand and sales data (Nahmias 1994). For all systems where there is a risk of stockout, there is a difference between sales and demand data. When demand occurs during stockout, customers have the option of waiting for the next delivery (backorder) or not buying from the specific retailer (lost sale). Obtaining the true demand data is often more difficult and not always possible compared to sales data, which is easily accessible. However, there exist several methods to approximate demand data (Nahmias 1994).

3.2 Markov Decision Process - MDP

One way to approach inventory management problems is by using Markov models (Axsäter 2015). With this framework, the problem is split into states and actions governed by the Markov property. The Markov property dictates that the future state of a system depends only on its current state, not its history (Sutton et al. 2018). In an inventory context, this means that the next state (e.g., Inventory Level and Inventory Position) is determined solely by the present state and the current action such as whether to place an order and in what quantity. While a Markov Chain observes the system's transitions passively, a Markov Decision Process (MDP) allows an agent to influence those transitions through deliberate actions. Consequently, when the goal is to optimize or improve an inventory policy, the MDP framework is typically more appropriate.

A Markov Decision Process (MDP) provides a framework for modeling decision making in situations where the outcomes are partly random and partly under the control of an agent (decision-maker) (Sutton et al. 2018). At each discrete time step, the process occupies a state $s_t \in S$. The agent chooses an action $a \in A$, which results in two things, the system moves to a new state s_{t+1} according to a probability distribution $P(s_{t+1}|s_t, a)$, and the agent receives a corresponding reward $R(s_t, a, s_{t+1})$. The ultimate objective is to find a policy that maximizes the cumulative expected reward over time.

One method of solving MDPs is called Dynamic Programming (DP). DP is one of the greatest influences in the formulation of RL algorithms (Sutton et al. 2018). In DP, there are two main ways to address MDPs, either using policy iteration or value iteration. Policy iteration is based on iteratively performing policy evaluation and policy improvement until an optimal policy is found. Value iteration is a form of truncated policy iteration that, instead of policy evaluation and improvement, uses the Bellman optimality equation to find an approximation of the optimal policy.

3.3 Machine Learning and Deep Learning

The exact definition of Machine Learning (ML) varies for different publications. One of the earlier definitions, given by Samuel (1959), is "the field of study that gives computers the ability to learn without being explicitly programmed". In modern applications of ML, neural networks or Artificial Neural Networks (ANNs) is the most common approach (Wang 2025). These networks use nodes to estimate a number of output parameters, based on a limited number of input parameters. These connections use different weights to alter the values between nodes and layers. For simpler networks, the input nodes are directly connected to the output nodes. In more complex settings, one or more "hidden layers" with nodes are added, where the input values are altered with an activation function in each node. These activation functions are preferably relatively simple, non-linear, differentiable functions, for example the sigmoid function. By adding "hidden layers", the algorithm becomes what is called "deep learning", a subset of machine learning.

The ANN learning process focuses on iteratively updating the weights that connect the nodes (Wang 2025). The update is typically achieved through the gradient descent algorithm, where the gradient of a loss function is calculated, which quantifies the discrepancy between the network's prediction and the actual target. Using backpropagation, the weights are adjusted in the directions that are expected to experience the largest reduction in loss. The weight update is made in steps of α (learning rate) chosen manually, where a high learning rate increases learning speed but poses the risk of being unstable.

To ensure that the model does not overfit the samples, the dataset is usually split into a training set and a separate test set. The loss function is calculated based on the training set, and performance on the test set is therefore often a more accurate prediction of model performance on new, unseen data.

3.3.1 Important Concepts in Reinforcement Learning

As mentioned in Section 1.1, RL is based on maximizing the expected reward. The algorithms for doing so are based on a few typical design specifications (Boute et al. 2022). Some of the more influential specifications are on-policy vs off-policy, model free vs model base, exploration vs exploitation, and temporal difference (Sutton et al. 2018).

Temporal difference (TD) learning is a central idea in RL. TD methods learn directly from experience through interaction with the environment without requiring a model of the environment's dynamics. TD methods continuously update estimates between time steps t without waiting for a final result. TD-update is performed according to Equation 3. Learning and updates are driven by the TD error $[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$. This error measures the difference between a current estimate of the value in state S_t and a more accurate estimate

that incorporates the actual reward (R_{t+1}) received in the next time step $t+1$. The error is smoothed with the factor γ and the update of the value $V(s_t)$ is done with the learning rate α. This approach is particularly effective for tasks that are continuous or have long episodes.

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \tag{3}$$

Exploration vs Exploitation An important consideration in RL is the balance between exploration and exploitation (Sutton et al. 2018). Exploitation is when the agent exploits what has already been learned and makes the decision based on what is expected to yield the highest return. Exploration, on the other hand, is based on the agent exploring the environment to increase knowledge. Exploitation lets the agent learn about the best policies and refine them by updating the expected reward, thereby accurately estimating the reward of a certain path. For exploration, the agent takes a random action. This random decision is motivated by increased knowledge about the environment. A lack of exploration may result in the agent getting stuck in local minima due to a lack of knowledge. A balance between exploration and exploitation is epsilon greedy agents ($0 < \epsilon < 1$). These agents explore with the probability of ϵ and exploit with the probability of $1 - \epsilon$.

The advantage of ϵ-greedy agents varies for different environments. Especially agents in environments with high reward variance may benefit from a high exploration to accurately estimate expected rewards (Sutton et al. 2018).

Model free vs Model based The primary distinction between model free and model based RL is if the agent uses a model of the environment (Sutton et al. 2018). The model aims to mimic the behavior of the environment, and thus can predict possible future situations. For stochastic environments (such as inventory management), a model could, for example, provide all possible future states with their probabilities (distribution model) or one state sampled from the distribution (sample model). In general, the advantages of a distribution model are greater, while a sample model is easier to obtain. Model free RL, learns through trial-and-error and do not plan over future states in the same manner as model based. Model based planning can improve learning in RL and increase sample efficiency compared to model free. However, advantages with model based RL rely on accurate environment models. For complex problems, model-free methods may be more advantageous if the construction of an accurate model is cumbersome or becomes a bottleneck in the problem solving.

On-policy vs Off-policy The difference between on-policy and off-policy in RL is based on whether the agent learns from its own behavior. In on-policy methods, the agent learns and improves the policy that makes the decision, essentially the agent learns only from the results of its own actions (Sutton et al. 2018). The off-policy methods focus on allowing the agent to learn about

a "target policy" using experience generated by a different "behavioral policy" (Sutton et al. 2018). This allows the agent to learn from external impact such as demonstration and random action. This difference makes on-policy algorithms more stable, while off-policy algorithms are more sample efficient.

Value based, policy based and actor-critic models RL-models are either based on value iteration (value based models) or policy iteration (policy based models). Furthermore, the class of actor-critic models combines the two. Value based models, just like value iteration, do not search for an optimal policy but rather an approximation of the optimal value function (Boute et al. 2022). Value-based methods of RL are based on bootstrapping and many of them are TD methods (Sutton et al. 2018). These methods have various advantages, for example, decreased variance and learning time. Policy based models directly approximate the optimal policy. Policy-based methods often use stochastic policies, in comparison to the deterministic policies used by value-based methods, and have better convergence properties than value-based methods. On the other hand, since policy-based methods are on-policy, they are very sample inefficient. When a sample is used, it cannot be used again. Furthermore, policy-based methods have higher variance, which can result in unstable training.

The hybrid approach of actor-critic models contains one network (the actor) that chooses policies that the second network (the critic) then evaluates by returning an expected value for the chosen policy (Boute et al. 2022). By using both value- and policy-based methods, the model is able to take advantage of both methods. The critic adds a value-based baseline. By adding a value-based critic to the policy-based actor, the problems and disadvantages of policy-based methods are mitigated. On one hand, this will increase the stability of the model; on the other hand, it comes at the expense of added computational requirements (Boute et al. 2022).

3.3.2 Teacher Policy

Since an RL agent learns through trial and error, it can be a cumbersome process that involves many wrong steps. This is a general drawback of RL and intuitively, if guidance could be applied to the autonomous learning process of an agent, it would not only perform better, but also increase learning speed.

Teacher-student framework One way of using a teacher policy is the teacher-student framework (Torrey & Taylor 2013). The teacher-student framework focuses on the use of RL agents as teacher agents to accelerate training of a second agent by giving it advice. The teacher agent has already learned an effective policy for the domain and task and can thus guide the student agent towards a good solution.

Torrey & Taylor (2013) propose four different teaching application algorithms: *early advising*, *importance advising*, *mistake correcting* and *predictive advising*.

Using *early advising*, the teacher agent only provides advice early in the training process of the student agent. Using *importance advising*, the teacher evaluates its advice based on the current state of the student agent. If the importance of advice in a state is over a defined threshold, advice will be given. *Mistake correcting* enhances importance advising by only providing the importance advising to the student agent if it intends to take the wrong action. *Predictive advising* adds yet another layer where the teacher agent predicts the student agents intended action a and then applying *mistake correcting* to this action.

The teacher-student framework is developed with agent to agent learning in focus but could also be applied in other settings where the teacher or the student is not an RL agent (Torrey & Taylor 2013). For inventory management, this could for example be the use of an analytically optimized (R,Q)-policy as the teacher.

Experience Replay Another way of using teacher policies, introduced by **?**, is with experience replay, where an agent stores previous encountered experiences (transitions between two states), which are then used again in training. The experiences are stored as tuples of transition information $e_t = (s_t, a, R, s_{t+1})$ in a data set $D_t = \{e_1, ..., e_t\}$. The experiences are then used by drawing samples $(s_t, a, R, s_{t+1}) \sim U(D)$ uniformly to a training batch for each training epoch. **?** does not only present the framework for experience replay, but also the use of teacher experience replay. The mechanism of teacher experience replay is similar to experience replay, but instead based on the teacher experiences. Experience replay has since been further explored, with results such as Hindsight Experience Replay (HER) and Prioritized Experience Replay (PER) being two seminal discoveries (Andrychowicz et al. 2017, Schaul et al. 2015).

Hindsight Experience Replay HER HER implementation provides agents with the ability of learning from mistakes (Andrychowicz et al. 2017). It was originally developed for robotics, a field in RL with sparse binary rewards for each goal state g. With standard experience replay, actions that result in a state $s \neq g$, mistakes, provide little information in environments with sparse binary rewards. To handle this, Andrychowicz et al. (2017) added a second transition tuple for each action, featuring a new goal $g' = s$. Thus, the action resulting in a mistake is now the correct action for g'. In this way, the agent stores the correct action for the goal g' in the experience buffer, which it can train on in a later batch.

Prioritized Experience Replay PER Compared to standard experience replay, PER does not sample experiences uniformly for replay (Schaul et al. 2015). PER uses the TD error (δ) to prioritize experiences. Experiences with a high TD error are sampled more frequently from the experience buffer, consequently providing the agent with prioritized training from unexpected results.

To prioritize on TD error, stochastic prioritization is used (Schaul et al. 2015). The probability of sampling transition i is defined as

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{4}$$

where p_i is the priority of transition i which is divided by the sum of all priorities p_k. Moreover, α is the prioritization coefficient, scaling the level of prioritization used. $\alpha = 0$ corresponds to uniform sampling. The priority of transition i is derived from

$$p_i = \frac{1}{\text{rank}(i)} \tag{5}$$

where $\text{rank}(i)$ is the rank of transition i when the transitions in the replay buffer are sorted according to the absolute value of the TD error of each transition i, $|\delta_i|$. New transitions in the replay buffer do not have a known TD error and are set to maximum priority to ensure that each transition is replayed once.

As prioritized replay changes the distribution of the samples used for training, it introduces bias that changes the solution to which the agent converges (Schaul et al. 2015). This bias is corrected by using importance-sampling (IS) weights, w_i.

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{6}$$

where N is the size of the replay buffer and β is the bias correction. If $\beta = 1$ full compensation for non-uniform probabilities, $P(i)$, is made. w_i is then incorporated into the model by using $w_i\delta_i$ instead of δ_i. Weights are normalized by $\frac{1}{\max_i w_i}$, thus only scaling updates downward.

Since the unbiased nature of updates is most important close to convergence at the end of training, the bias correction β is linearly increased during training using a beta increment, $\beta_{\text{increment}}$, from the initial value β_0 to 1 (Schaul et al. 2015).

3.4 Q-learning

A common value-based algorithm in RL is Q-learning (Sutton et al. 2018). Q-learning was one of the early breakthroughs and is an algorithm that aims to approximate the action-value function (q_*) based on an MDP (Watkins 1989). The action-value function gives the expected value for each legal action in every state, given that the action with the highest expected value is taken in each following state. Q-learning is an off-policy TD (Temporal Difference) control algorithm (Sutton et al. 2018). The approximation of q_* is denoted Q and is calculated according to Equation 8, using the target function Y_t calculated according to Equation 7. The variables are explained in Table 2. Under the assumption that all state-action pairs are continuously updated, Q can be shown

to converge to q_* (Sutton et al. 2018). This assumption holds for all ϵ-greedy agents.

$$Y_t \equiv R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \tag{7}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[Y_t - Q(s_t, a_t)\Big] \tag{8}$$

Table 2: Explained variables for Equations 7 & 8

s_t	Current state
s_{t+1}	The next state the agent moves to
a_t	Action taken by the agent
a_{t+1}	Next action taken by the agent
R_{t+1}	Received reward for taking action a_t in state s_t
γ	Discount factor for future reward
α	Learning rate regulating the impact of new information

The action-value function aims to estimate all future expected rewards for each legal action in each possible state (Sutton et al. 2018). This is saved in a Q-table that is continuously updated with new values of $Q(S_t, A_t)$ according to Equation 8. Q-learning algorithms focus on choosing the highest expected return for each state in which the agent finds itself.

3.4.1 Deep Q Network (DQN)

While non-deep Q-learning can be very effective for problems with a tractable Q-matrix, it is quickly rendered unusable when the state and action spaces increase (Sutton et al. 2018). To circumvent the issue of exploding state and action spaces, DQN is used.

An RL agent using DQN leverages benefits seen for neural networks in Q-learning. In comparison to non-deep Q-learning, DQN does not use a Q-matrix to store all state and action values; instead it estimates the value of action a in state s using a neural network. State information is fed into the neural network as the input layer. This information is then passed through one or multiple hidden layers to the output layer where it is formulated as an expected Q-value for the observed state s_t and taken action a_t. Figure 1 visualizes the system architecture. The Q-value is updated according to Equation 8, as for non-deep Q-learning. Since the DQN is based on a neural network, the weights are updated using backpropagation and gradient decent (Boute et al. 2022).

Different neural network structures can be used in a DQN, and their respective use-cases will impact the use-cases for the DQN. For example, a convolutional neural network is well suited for handling image or video inputs while a fully

connected feed forward neural network is better suited for tabular tasks without spatial dependencies. Thus, their respective DQNs will inherit these capabilities.

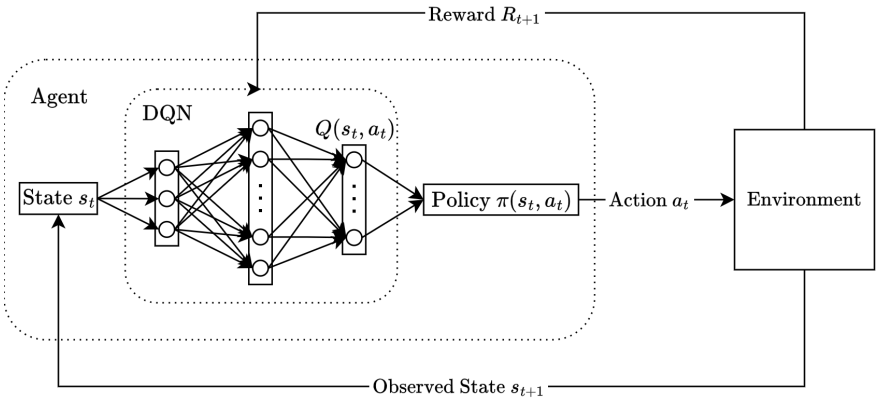

Figure 1: Visualization of a DQN-architecture with one hidden layer

3.4.2 Double Deep Q-network (DDQN)

Although DQNs are a seminal approach to solving certain cases of MDPs, they are not without issues. Firstly, while it can work well in practice, there is no theoretical proof of convergence for the DQN algorithm (Oroojlooyjadid et al. 2022). Therefore, DQN training may not be very stable (**?**). At times, instead of converging toward an optimal solution, the loss function can diverge. Furthermore, in standard DQNs, the target values and estimates originate from the same function and thus, the function is used to update itself. This can lead to a so called unstable target function. Secondly, standard DQNs are known to overestimate the Q-value, negatively affecting training (Van Hasselt et al. 2016). One proposed solution to this, a Double Deep Q-Network (DDQN), decouples the target values and estimation values over time using two networks. The target network is periodically copied from the evaluation network. This stabilizes training and mitigating the overestimation bias.

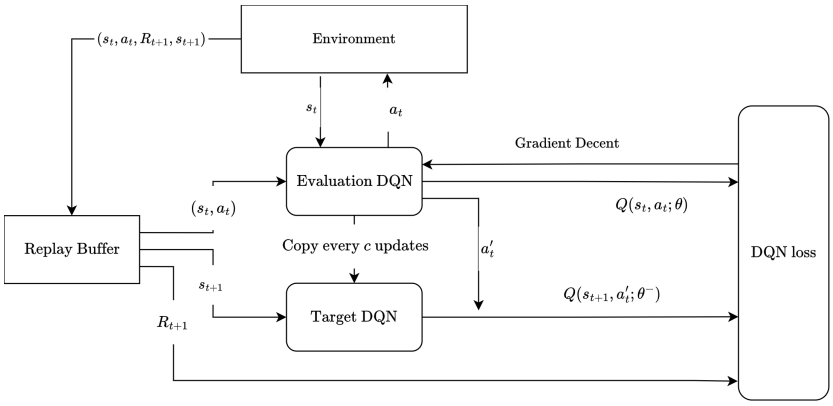

Figure 2: The architecture of a Double DQN (Van Hasselt et al. 2016). See Figure 1 for DQN architecture.

In a Double DQN, an evaluation network and a target network are trained in parallel. See Figure 2 for the network structure. The evaluation network is updated every iteration, just like a standard DQN, while the target network is copied from the evaluation network every c iteration. For each training round, the target Q-value and evaluation Q-value are compared using Mean Squared Error (MSE) loss. The MSE is then used in backpropagation to update the evaluation network. In this way, overestimation of the Q-value is avoided. The target value $Y_t^{\text{Double DQN}}$ in Double DQN is calculated with

$$Y_t^{\text{Double DQN}} \equiv R_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t); \theta_t^-), \tag{9}$$

where θ_t is a parameter that describes the evaluation network and θ_t^- is a parameter that describes the target network. The greedy action for $Y_t^{\text{Double DQN}}$, denoted a'_t in Figure 2, is found using the evaluation network, as seen in Equation 9. $Y_t^{\text{Double DQN}}$ is used to calculate the DQN loss and update the parameters of the evaluation network.

3.4.3 Dueling Deep Q-network

When taking an action, for example to order or not to order in an inventory management system, there may in some states be a negligible difference between actions (Wang et al. 2016). For these states, it is interesting to learn not the state action combination, as in a standard DQN, but rather the value of the state itself (since the action is of less importance). Furthermore, small differences are hard to separate for neural networks. Thus, for states with many similarly valued actions, a clearer distinction between actions could be beneficial. The architecture proposed by Wang et al. (2016), called Dueling DQN, separates the state and action values and consequently decouples the value of a state from its actions. Figure 3 shows the architecture proposed by Wang et al. (2016).

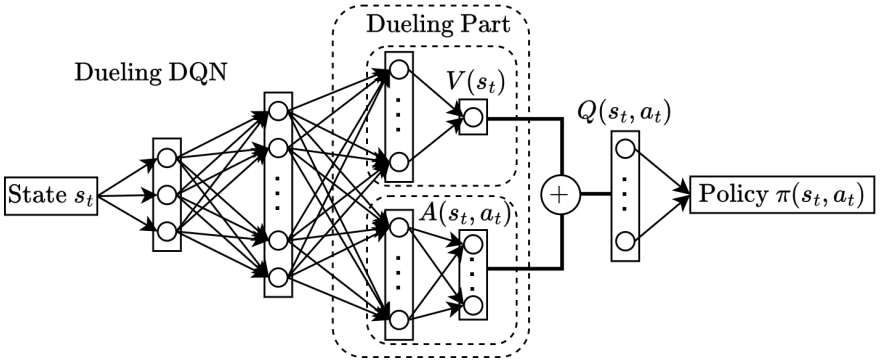

Figure 3: The architecture of a Dueling DQN

Except for the output layers, the network is constructed similarly to a simple DQN (Wang et al. 2016). In the output layers, the network is split into two parallel streams, one value stream and one advantage stream. The value stream estimates the value of the state s_t, while the advantage stream estimates the advantage $A(s_t, a_t)$ of the action a_t compared to all other legal actions a'_s in the state s_t. This is done for every action in the state s_t, and thus the value stream outputs a vector corresponding to the number of actions in s_t. To transform the value $V(s_t)$ and the advantage $A(s_t, a_t)$ to a Q-value, the Q-value $Q(s_t, a_t)$ is updated using Equation 10 where $|\mathcal{A}_s|$ is the number of possible actions in state s_t. Thus, the Q-value $Q(s_t, a_t)$ is calculated from the value $V(s_t)$ and the deviation of advantage $A(s_t, a_t)$ from the mean of advantages for legal actions a'_s in state s_t. The policy is then found by maximizing the Q-value in s_t according to Equation 10.

$$Q(s_t, a_t) = V(s_t) + A(s_t, a_t) - \frac{1}{|\mathcal{A}_s|} \sum_{a'_s \in \mathcal{A}_s} A(s_t, a'_s) \tag{10}$$

A dueling DQN architecture allocates more resources toward the estimation of state values in comparison to a simple DQN (Wang et al. 2016). Accurate state value estimations are essential for TD based methods such as Q-learning, and therefore a dueling DQN is found to be capable of providing a better and more robust policy evaluation (Sutton et al. 2018, Wang et al. 2016).

3.4.4 Dueling Double Deep Q-network (DDDQN)

Since a double DQN and a dueling DQN aim to solve different issues with a DQN, they can be combined into a Dueling Double Deep Q-network (DDDQN) to receive synergetic benefits (Wang et al. 2016). This is done by switching the main and target DQNs in Figure 2 for the dueling DQN architecture, Figure 3.

3.5 Policy Gradient Methods

Policy gradient methods are methods that instead of using gradient descent calculated from a loss function, as is customary in ANNs, update over gradient ascent (Sutton et al. 2018). The gradient estimator used for gradient ascent is obtained by differentiating the objective function L (Schulman et al. 2017).

3.5.1 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO), introduced by Schulman et al. (2017), is a model-free algorithm that can be used in both continuous and discrete action spaces. It is a further development of Trust Region Policy Optimization (TRPO). TRPO, while effective, is computationally heavy with complex implementation. PPO is based on approximations of the TRPO methods.

PPO has three key concepts: (1) an actor-critic architecture, (2) an advantage function, and (3) a clipped surrogate objective (Schulman et al. 2017).

The clipped surrogate objective is a loss function substitute that is applied to the policy-based actor. It uses a clipping mechanism to the loss function to ensure that policy updates are made in a conservative fashion. The clipping mechanism introduced by Schulman et al. (2017) limits positive policy updates to a maximum value. For negative updates, the clipping is performed close to zero, preventing the loss function from adopting values between the clipping limit and zero, thus ensuring a minimum penalty size.

3.5.2 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is based on Deterministic Policy Gradient (DPG) (Lillicrap et al. 2020). It is an off-policy, actor-critic approach. In the actor, the DPG algorithm is used to deterministically map states to a specific action with a parameterized actor function. The critic utilizes Bellman equations to learn the Q-value $Q(s_t, a_t)$ as in Q-learning. DDPG is similar to the use of a target network with a soft update. Furthermore, DDPG apply batch normalization to scale features across environments and units.

In contrast to DQN, DDPG is modified for use in continuous action spaces. In continuous action spaces, exploration is more difficult since the policy itself is deterministic. To ensure exploration is performed across a wide variety of actions and states, noise is added during training.

4 Existing RL-model - a deep dive

Syncron has performed initial testing of RL on inventory optimization. The initial RL-model is modeled for a single-echelon system with respect to only one product and using a non-deep Q-learning algorithm. The inventory management

of the model handled only the decision if to order a fixed order quantity of 2. The simulated demand pattern reoccurred weekly over a period of 52 weeks. The distribution of demand is presented in Appendix 10. This model was found to outperform an (R,Q)-policy when evaluated using total cost derived from stockholding and backorder costs as key performance indicators. Additionally, the model assumed a constant lead time without disruptions in the supply chain.

4.1 Cost and service level approximation

To account for a desired service level, the parameter α was introduced. α was defined as the quota between backorder and holding cost ($\alpha = B/H$). Additionally, α approximates the fill rate by Equation 11. While this approximation does not guarantee fulfilled service levels, it uses induced costs which implicates similar constraints. The main advantage with α it that it enables a simplified version of cost function, Equation 12. This cost function gives a different interpretation of the accumulated cost than using backorder and holding cost separately, Equation 13 for comparison. However, the relationship between different policies remain proportional and a comparison of cost change can accurately be made. Additionally, the usage of α reduces the number of parameters from 2 to 1 and thereby simplifies the calculations when varying parameters. The total cost (TC) in Equations 12 and 13 have different interpretations but remain comparable.

$$\alpha/(\alpha + 1) \approx SL \tag{11}$$

$$TC = \text{Avg (daily) Stock on Hand} + \alpha \cdot \text{Avg (daily) Backorders} \tag{12}$$

$$TC = H \cdot \text{Avg (daily) Stock on Hand} + B \cdot \text{Avg (daily) Backorders} \tag{13}$$

4.2 Syncron's initial RL-model

The initial RL-model uses the Q-learning algorithm to find a solution. The hyperparameters presented in Table 3 are used.

Table 3: Hyperparameters from the initial RL-model

Learning rate (α)	0.1
Discount factor (γ)	1
Epsilon	0.01

In addition to hyperparameters, some limitations were set to guide it in a relevant direction. In addition to the rewards representing the holding and backorder cost, a reward was set at -1 000 000 if the inventory level went outside the limit of -10 to 10. This also truncated the episode to avoid training in

"obviously" bad directions. To further increase agent exploration in the initial stages, an initial optimistic value of 100 is set for every Q-value. This is not a realistic approximation, but it will converge to the actual Q-value (Sutton et al. 2018). An advantage with this choice is that the unexplored state-actions will be explored and contribute to earlier convergence for the action-value function.

Action space
Since the model had a fixed order quantity of 2, the action space was binary with only the decision to order or not.

State space
The state space for the RL-model was varied by changing how many days of historic demand the agent could access. In addition, the state always contained information on the inventory level (IL) and the number of incoming orders. The memory of historic demand was varied from 0 to 6 and contained a binary variable for each day recording if customer demand happened during a specific day. Increasing the memory further was unnecessary, since demand was reoccurring weekly.

4.3 Comparison with (R,Q)-policy

In order to compare the results from the RL-policy with results from an (R,Q)-policy, it is desirable to find the optimal (R,Q)-policy. The chosen method for finding such policy was through simulation over all relevant order-levels (R varied from -4 to 8) while $Q = 2$. To account for different service levels, α was varied according to Table 4. The optimal order level (R) for each α, together with the achieved and estimated service level, is presented in Table 4. Estimated service levels are based on α and Equation 11. It could be ensured that, for this simulation, an R outside the tested range (-4 to 8) is not optimal or relevant to test. This follows from analysis of the backorders. For $R \leq -2$, all orders are placed as backorders and the system never hold any stock. A further decrease in R, therefore gives increased backorders without decreasing inventory level, only resulting in an increased cost. For $R \geq 6$, all demand is met from stock (SL=100%). A further increase would therefore increase stock without decreasing backorders and solely increase cost. Similarly to the RL-policy, the lead time was 1 day.

Table 4: Estimated and achieved service levels for different values of α in optimal R.

α	**Order level (R)**	**Estimated SL**	**Achieved SL**
2	2	67%	61%
3	3	75%	72%
5	4	83%	83%
9	5	90%	94%
19	6	95%	100%

4.4 Comparison between results

For all variations of state space in the RL-model, the resulting ordering pattern both varied and improved from the (R,Q)-policy. For the most simple state space without memory of historic demand, the RL-model reached a cost reduction between 12.3% ($\alpha = 2$) and 26.7% ($\alpha = 19$). Both policies prevented stockouts completely. Thus, the full cost reduction achieved by the RL-model is from reduced stock on hand. The ordering patterns is repeating with a two week interval and is presented in Appendix 10. For larger state spaces, the solutions got even better. The cost reduction increased up to the range of 56.7% ($\alpha = 2$) to 64.0% ($\alpha = 19$).

The increased cost reduction has a high risk of being caused by overtraining. Thus, for stochastic demand increasing the state space with states of historic demand may result in far less improvement. Despite this, the results suggest that expanding the state space can improve the accuracy of ordering policies.

5 Literature Review

RL implementation on spare parts inventory management is in its cradle. Following a limited amount of research on the implementation of RL in inventory management for spare parts, an extended review of RL in inventory management will be made. In addition, some systematic reviews of articles have been published and studied in this review to improve understanding of the landscape and what methods are common. However, publications on machine learning and inventory management have increased, especially since 2015 (Bergsma et al. 2025). It can also be concluded that the increase is combined with a growing influence of the private sector (Ahmed et al. 2023). This influence is measured in a wide range of aspects, such as share of private authors and co-authors of published articles, share of PhDs hired by industry, and percent of largest AI models that are from industry.

As shown by Bergsma et al. (2025), a large part of RL research on inventory management applies simplified versions of model assumptions, especially with respect to the number of items considered and the lead time. Furthermore, there is a high potential for improvement when applying RL-models to complex inventory management systems. These models could for example consider several number of items simultaneously. Although the complexity considered in the RL-model is often simplified with respect to the inventory management problem, 83% of all RL articles reviewed by Bergsma et al. (2025) use Deep Reinforcement Learning (DRL).

Table 5 presents the complexity considered in the 51 articles reviewed by Bergsma et al. (2025) that used RL. It can be noted that not all 51 articles regarded all aspects, which is why the simple and complex articles do not always sum up to 51. The exact classification between simple and complex is varied for different aspects and is described in Table 5.

Table 5: Comparison between simple and complex configurations (Bergsma et al. 2025)

	Simple		Complex	
	Description	Value	Value	Description
Items	One item	48	3	Multiple items
Supply process	$L = 0$	23	28	$L \geq 1$ or stochastic
Procurement	No order cost	31	19	Order cost considered
Backorder	Backorders	24	25	Lost sales
Shelf life	Unlimited	40	10	Limited
Echelon	single-echelon	24	27	multi-echelon
Capacity constraint	Unlimited storage	35	16	Limited storage

5.1 RL application on spare parts inventory management

Five relevant articles were found on RL implementation on spare parts inventory management. Of the five articles, two (Wang et al. 2025, Zheng et al. 2024) handle the joint optimization problem of maintenance and inventory management while three (Yu et al. 2020, Zhou et al. 2024, van der Haar et al. 2026) propose models to solve the isolated problem of inventory management of spare parts; see Table 6.

Table 6: The five articles, their inclusion of maintenance, and which ML-algorithms they applied

Source	**Maintenance**	**ML-Algorithm**
Wang et al. (2025)	Yes	LSTM & PPO/DDQN
Zheng et al. (2024)	Yes	HDRL (DDDQN)
Yu et al. (2020)	No	DDDQN
Zhou et al. (2024)	No	EM-VDTD3 (DDPG)
van der Haar et al. (2026)	No	DCL

Since the studies outlined in this section all propose solutions to spare parts inventory management, they are explored in depth to provide a picture of the current, limited, research landscape and which models and approaches are viewed as state of the art.

5.1.1 Combined Maintenance and Inventory Management

Both Wang et al. (2025) and Zheng et al. (2024) investigate the inventory management of spare parts as a sub-problem in the joint optimization of maintenance and spare parts inventory. Both articles highlight the possibilities of condition-based maintenance with the introduction of intelligent sensors and real time monitoring data. Moreover, the claim is made that sequential study of spare parts and maintenance cannot ensure a holistically lowest cost. However, this complicates inventory management. More complex maintenance planning introduces additional uncertainties, which in turn could lead to increased risk exposure for downstream inventory management. Consequently, as stated by both Wang et al. (2025) and Zheng et al. (2024), this requires computationally efficient algorithms to handle large state and action spaces.

Wang et al. (2025) proposes a sequential setup using a Long Short Term Memory (LSTM) algorithm for maintenance prediction combined with a Proximal Policy Optimization (PPO) algorithm for inventory optimization. Zheng et al. (2024) on the other hand, develops a Hybrid Deep Reinforcement Learning (HDRL) algorithm based on a Dueling Double Deep Q-Network (DDDQN) structure.

Sequential LSTM & RL-model as presented by Wang et al. (2025)
Wang et al. (2025) proposes an integrated maintenance and inventory optimiza-

tion architecture for power system asset management.

Prediction of the Remaining Useful Lifetime (RUL) is performed using a combination of an LSTM model and a hybrid Wiener process (Wang et al. 2025). An LSTM is a further development of the structure of an ANN (Wang 2025). Instead of a simple feedforward structure, it incorporates a memory property. It excels at forecasting due to its memory property which enables temporal pattern recognition. The hybrid Wiener process simulates the physical degradation process of spare parts in use (Wang et al. 2025). The hybrid Wiener process accounts for both linear drift and stochastic shocks. A Wiener process is a continuous-time stochastic process with a distribution $W_t \sim \mathcal{N}(0, \sigma^2 t)$, where σ^2 is the process variance and t time units. It acts as a mathematical model for random continuous motion. Moreover, it is modified to introduce linear drift (using a mean $\mu \neq 0$) to the process. The LSTM and Wiener process are combined, achieving synergetic advantages in terms of interpretability and predictive accuracy for the forecast.

The forecast uncertainty in the input to the RL-model is handled using Distributionally Robust Optimization (DRO), also known as Wasserstein ambiguity sets. DRO allows the model to account for the possibility of an incorrect forecast by instead using the worst case scenario from a set of distributions within a set distance from the forecast distribution. In this case, the distance is calculated using the Wasserstein distance. The Wasserstein distance is a measurement of the closeness between distributions based on the sum of the relative divergence between two distributions in each point. Using DRO, the estimation is hedged against a forecast distribution that does not accurately represent the data. This increases the robustness of the policy. The RL-model interacts with a custom simulation environment built on top of the degradation and inventory dynamics. There, it learns parameterized scheduling policies. The policy learned is enhanced using surrogate modeling. Surrogate modeling uses gaussian process regression to approximate the loss landscape. This allows for increased computational efficiency in the iterated calculations of loss in the RL-model. The spare parts inventory is optimized using total cost, system reliability under resource constraints, and resilience (worst-case). For a holistic end result, the prediction model is co-trained with the inventory model, thus decreasing end-to-end system cost. The recursive loss function used as the objective for the RL-model is formulated to work with both PPO and DDQN.

Using a dual layer architecture like this allows for capturing nonlinear temporal features from the historical data while providing a probabilistic capturing of the health status of the asset. Combining LSTM with traditional inventory models, like the (s,S)-policy, often sees multiple restrictions in multi-asset uncertain environment, thus limiting tractability. Compared to a heuristic policy, the RL-policy shows a more context-aware behavior that varies inventory dispatch depending on the inventory position and the criticality of the asset.

Through a case study based on an energy network consisting of 75 assets (in this case wind turbines) and 4 regional depots over a 24-month planning horizon, the proposed framework was validated across multiple performance dimensions. The case study was based on publicly available data. Compared to traditional periodic and perfect-forecast baselines, the integrated model achieved a 34% reduction in total cost, a 72% reduction in emergency interventions and consistently maintained more than 90% asset protection coverage. The case study incorporated a DDQN as the RL-model.

HDRL as presented by Zheng et al. (2024)
Zheng et al. (2024) states that all previous work on joint spare parts and maintenance optimization implements solutions that are only applicable to low-dimensional problem scenarios. Earlier work has used Q-learning for MDP:s, but Zheng et al. (2024) adopts a Dueling Double Deep Q-Network (DDDQN) structure.

The DDDQN-model is constrained by the available quantity and service level for each supplier and location, respectively. In addition, the model adopts the assumptions of independent deterioration of components, perfect maintenance, instant replacement, and nonexistent supply risks. The DDDQN-model is combined with a rule-based maintenance strategy, the "greedy strategy", and a threshold control algorithm developed for multiple suppliers, which sets the initial solution of the inventory management system. Thus, the model is named HDRL rather than a straightforward DDDQN-model.

The "greedy strategy" is adopted by Zheng et al. (2024) in regards to maintenance and is used in the early stages of training to improve the solving efficiency of the HDRL algorithm. The "greedy strategy" divides the component states into three categories: good, intermediate, and poor. Replacement strategies are then based on the component state categories. Furthermore, the initial solution of the inventory management system is set using an improved threshold control algorithm. A threshold control algorithm is based on ordering at a set threshold. In contrast to traditional threshold control algorithms, like the (s,S)-policy, the improved threshold control algorithm is suitable for a system with multiple suppliers.

The HDRL algorithm is compared to the value iteration algorithm. Value iteration returns optimal solutions to the MDP but with exponentially increasing solution time when increasing the complexity of the system. For Zheng et al. (2024), the solution time for the value iteration algorithm exceeds one week when the number of components is ≥ 5. The HDRL algorithm, on the other hand, has a much slower increase in solving time. For 5 components, it is solved in 5.6 h. The average cost gap between HDRL and value iteration is 3.86%, and when the HDRL algorithm is used for systems with more than 2 suppliers and more than 3 components, the solving time can be reduced by at least 95.99%.

Furthermore, the HDRL algorithm is compared to a regular DRL algorithm. The HDRL performs better in terms of cost, showing that the introduction of the initial threshold control solution and the greedy strategy can effectively improve performance. In addition, the HDRL algorithm shows faster convergence than the DRL algorithm.

5.1.2 Spare parts inventory management in focus

Yu et al. (2020) and Zhou et al. (2024) propose DRL-models as a solution for spare parts inventory management. Yu et al. (2020) proposes a multi-agent reinforcement learning method based on the DDDQN structure to solve a two echelon spare parts inventory management problem. Zhou et al. (2024) on the other hand, develops an RL-model called EM-VDTD3 (Experience buffer Modification - Value Decomposition Twin Delayed Deep Deterministic policy gradient).

DDDQN with periodic review as presented by Yu et al. (2020)
Yu et al. (2020) proposes an RL based framework to find a good solution to a single product, two echelon spare parts inventory management problem. The inventory system consists of one central and 3 local warehouses. The system includes transshipments and uses periodic review. The proposed solution is based on the DDDQN framework described earlier.

Compared to classic Double DQN that uses a fixed update interval for the target network, Yu et al. (2020) adopts a soft update per Equation 14. θ denotes a parameter for the evaluation Q-network and θ^- denotes a parameter for the target Q-network. Moreover, τ is the share of the evaluation Q-network to be adopted by the target Q-network.

$$\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-, 0 \leq \tau \ll 1 \tag{14}$$

When the results are compared to a genetic algorithm (GA), an (s,S)-policy, it is found that the DDDQN reduced costs by 4% for the evaluated case (Yu et al. 2020). Furthermore, it was evident that the DDDQN was more restrictive in ordering to the central warehouse due to its holistic perspective in comparison to the GA.

EM-VDTD3 as presented by Zhou et al. (2024)
As Yu et al. (2020), Zhou et al. (2024) proposes a solution to a two echelon system with lateral transshipments and full information sharing using Multi Agent Deep Reinforcement Learning (MADRL) where each warehouse is represented by an agent. The inventory system on which the model is tested consists of 6 central and 36 local warehouses. Each central warehouse answers for 6 local warehouses. The supply chain is modeled using Dec-POMDP (decentralized partially observable Markov decision process). The proposed model is called

EM-VDTD3.

Experience Buffer Modification (EM) combines a teacher policy and an experience buffer. It can be viewed as a form of imitation initialization. In Zhou et al. (2024), the teacher policy used is an approximate genetic algorithm. The experience buffer is modified by initializing it with teacher experiences before the agent interacts with the environment. When training begins, the teacher policy is used in every d step and thus, every d step a teacher experience is stored in the buffer. Thus, teacher experiences are more influential early on in the training process when the buffer is filled with them. When the agent learns policy behavior with a good reward, its dependence on the teacher decreases. By initializing the model training with policy values that are expected to be closer to the optimal values, the model avoids non-convergence, which was found to be a problem for VDTD3. The approximate teacher policy performs a computationally efficient initialization of the experience buffer, compared to GA. The computation time of the approximate algorithm (0,4 s) is 22500 times smaller than that of the GA (9000 s).

Value Decomposition (VD) uses a team value function, a joint value function for all agents n, which is disaggregated and used in the model for each agent. This agent-wise function is defined as Equation 15 where a_{nt} is an action taken by agent n based on its state s_{nt}. Moreover, $\mathbf{s}_t$ denotes the global state and $\mathbf{a}_t$ is the joint action of the agents. The decomposition of a team value function into agent-wise functions not only correlates the agents to each other, retaining a holistic picture, but also reduces the number of network parameters. With a reduced number of network parameters the computational complexity is reduced, which increases scalability.

$$Q(\mathbf{s}_t, \mathbf{a}_t) \approx \sum_{n=1}^{N} Q_n(s_{nt}, a_{nt}) \tag{15}$$

TD3 is an extension of deep deterministic policy gradient (DDPG). A common problem when estimating value functions is that the value is estimated optimistically. A solution to this is to introduce a second critic network. The critic networks work in tandem, as "twins", and the smallest value function estimation is used as the "correct" value which is used to form the target function. The target function is in turn used to evaluate the loss function. Although updates in DDPG are soft updates, see Equation 14, TD3 utilizes a delayed update of the policy, updating it less frequently than the Q-function. Therefore, the risk of updating the policy with less stable instances is reduced. Furthermore, TD3 adds noise to the target action, thus smoothening out the Q-value of different actions. This reduces the ability of the policy to exploit Q-function errors.

It is important to note that TD3, while based on Q-learning, is quite different from DQN, the main difference being the action space. DQN is suitable for a discrete action space, TD3 assumes a continuous action space. This decreases the

number of output nodes in the actor network, which in turn improves scalability.

According to Zhou et al. (2024), EM-VDTD3 performed the best of the tested DRL algorithms. Furthermore, compared to VDTD3, which only converged 10% of the time, EM-VDTD3 always converged to a high-quality solution. GA was found to need data from $5 \cdot 10^7$ review intervals while DRL methods only needed simulation data from $5 \cdot 10^5$ review intervals. GA was thus seen as less data efficient than DRL methods.

5.1.3 Multi-echelon, multi-item with expected reward

van der Haar et al. (2026) takes on the challenge of optimizing an IM system with 10 000 different items, one central warehouse, and between 5 and 59 local warehouses. Using Deep Controlled Learning (DCL), a concept introduced by Temizöz et al. (2025), together with intelligent initialization and reward shaping, van der Haar et al. (2026) achieves between $3.8 - 5.1\%$ cost savings compared to a state of the art heuristic optimization method used at the case company. Furthermore, the model was found to decrease critical materials shortage by $> 10\%$.

The system allowed for both proactive and reactive transshipments between warehouses and used both real demand data from the case company and generated demand using a Poisson distribution. Generated demand was included in the study to increase the transparency of the result. All item demand were viewed as independent.

To allow for optimization using DRL in an industrial scale inventory system, van der Haar et al. (2026) states that new approaches are needed to effectively ensure convergence of the model and competitive results. To manage the inherent problems of DRL, three techniques were used and improved upon in combination with DCL; action space decomposition, DRL input and output standardization, and reward smoothing. The action space decomposition allowed for linear action space growth rather than exponential when increasing the size of the problem. To decrease the size of the action space, analytical evaluation of actions were combined with the RL agent. Only a subset of potential actions were then used as action space for the RL-model. Furthermore, it was evaluated if the use of the RL-model was necessary prior to each decision. Instead of using the actual reward of a transition, an estimate of the expected reward value was used. Since the real expected value was too cumbersome to calculate, estimates of the expected value were used. This decreased the reward variance, a general problem for DRL-model training.

van der Haar et al. (2026) is able to show how an implementation of DRL on an industrial scale inventory system could be conducted. By intelligent computations to calculate feasible actions rather than punishing the model, as well as more hands on guiding of the RL-model, they are able to achieve a groundbreaking result: a DRL-model that outperforms existing heuristics for industrial

scale systems with feasible computational requirements. The model has since been implemented in the case company algorithm for inventory management with successful results.

5.1.4 Data usage in RL for spare parts inventory management

The training data in research on RL for spare parts for inventory management vary between papers. Zheng et al. (2024) and Yu et al. (2020) used synthetic demand generated from a Poisson distribution, Yu et al. (2020) present their distribution parameters ($1 \leq \lambda \leq 3.33$) and Zheng et al. (2024) do not clearly state the parameters. Wang et al. (2025) apply a stochastic deterioration of parts modeled by a Poisson process with a μ between 0.2 and 0.4. Zhou et al. (2024) utilize stochastic order instances where demand is characterized by multi-product requirements and specific due dates, rather than a simple aggregate demand distribution. van der Haar et al. (2026) uses both real demand data from their case company and Poisson distributions based on the demand for each of the 10 000 items.

5.2 General RL implementation in inventory management

5.2.1 RL algorithms used in research

For RL in inventory management, the dominant algorithm to use is PPO, with DQN being the second most used algorithm. PPO is often favored for its ability to handle complex inventory situations following high stability and its ability to handle high-dimensional or continuous action spaces. For DQN, sample efficiency is a valued characteristic as well as a suitability for discrete action spaces (Bergsma et al. 2025). The reviewed articles with DRL implementation are presented in Table 7, with their algorithms and if the algorithms were modified or implemented as is.

Table 7: Algorithms in the reviewed articles

Article	Algorithm	modified
Zhou et al. (2026)	PPO	Yes
Yan et al. (2026)	Special	Yes
Rozhkov et al. (2025)	DQN, PPO, A2C	No
Wang & Li (2025)	DQN	Yes
Hu et al. (2025)	PPO	No
Temizöz et al. (2025)	DCL	Yes
Liu et al. (2025)	PPO	Yes
Smith et al. (2025)	PPO	No
Ilie & Semenescu (2025)	PPO	No
Yunxiang & Zhao (2024)	Q-learning	Yes
Batsis & Samothrakis (2024)	PPO	No
Burtea & Tsay (2024)	DQN	No
Stranieri et al. (2024)	PPO, A3C, PG	No
Kim et al. (2024)	DQN	No
Tian et al. (2024)	A2C, PPO	No
Saha & Rathore (2024)	DQN	No
Geevers et al. (2024)	PPO	No
Meisheri et al. (2022)	DQN, PPO	No
Wang et al. (2022)	DQN	No
Abu Zwaida et al. (2021)	DQN	Yes

As seen in Table 7 some implementations of RL in inventory management apply modified algorithms. These solutions are for example "Diffusion model with Entropy-guided Multi-Agent Proximal Policy Optimization" (DE-MAPPO) (Bergsma et al. 2025) and "Randomized Ensembled Double Q-learning" (REDQ) (Yunxiang & Zhao 2024).

Additionally, it should be noted that research applying more than one algorithm either implements split optimization such as decoupling forecasting and ordering (described in Section 5.2.3), or makes comparisons between algorithms. The exact result of the comparisons vary between settings where for instance Meisheri et al. (2022) finds that DQN marginally outperforms PPO and Stranieri et al. (2024) finds that PPO is superior to both A3C and PG. The decision between algorithms for research implementing one algorithm is generally motivated by the given settings for the inventory problem rather than one algorithm being viewed as superior.

5.2.2 Lack of data, emphasis on data usage and substitutes

As for all optimization of spare parts inventory management, lack of data and infrequent demand pose a problem, especially for RL where training data is essential for achieving desirable results. As stated by Bergsma et al. (2025), testing inventory management RL-models on real data is an area with limited

research. Handling the transfer from a simulated training environment to real applications is therefore an important part to consider. Some articles (Saha & Rathore 2024, Smith et al. 2025, Wang & Li 2025) conduct experiments with sufficient real training data to train the model only on real data. Other approaches used are meta-learning (Batsis & Samothrakis 2024), teacher policy (Yan et al. 2026) and transfer learning (Hu et al. 2025). Another common and simpler approach is to "first train agents in a simulated environment, which is usually constructed based on estimation with limited real data, and then transfer agents' policies to the reality or target domain" (Hu et al. 2025). However, this approach poses the risk of resulting in poor performance on real data if the simulated environment differs from the real environment.

Hu et al. (2025) mentions three sub-fields which aim at solving the generalization problem and applicability to real world, Robust RL (RRL), Transfer learning in RL (TRL) and Zero-Shot Generalization in RL (ZSG). RRL applies worst-case optimization of environmental errors and uncertainties that results in conservative policies (Roy et al. 2017). TRL uses knowledge about different environments with similarities to reduce the required data from the target domain (Zhuang et al. 2021) and ZSG creates robust policies which aim to be transferable to the target domain, but also that domain adaption is not applicable and solutions are more general (Kirk et al. 2023). These methods have not been widely tested on inventory management problems, but are more common in, for example, robotics.

Meta-learning is often described as the concept of learning how to learn (Batsis & Samothrakis 2024). The meta-learning model learns from the context and outcome of other ML-models and can, for example, focus on what is commonly changing between environments. In the application by (Batsis & Samothrakis 2024), this approach is implemented by first training separate reinforcement learning agents in multiple simulated inventory environments with varying demand patterns and structures. The decisions generated by these agents are then used to train a meta-learner that maps states and contextual parameters directly to ordering actions. By learning how optimal policies vary across environments, the model enables generalization to new products without training an RL-model from scratch.

5.2.3 Decoupling forecasting and ordering

The majority of applications of RL in inventory management are, as mentioned, considering a joint problem between forecasting and ordering. However, some papers consider these two problems separately (Smith et al. 2025, Meisheri et al. 2022, Tian et al. 2024, Wang & Li 2025). In contrast to research about forecasting completely decoupled from ordering, these papers handle inventory management simultaneously but with, for example, separate RL agents. Wang & Li (2025) uses a Convolutional Neural Network (CNN) to process historic demand and then implement non-deep Q-learning for ordering decisions. Tian et al.

(2024) implements an approach in which A2C and PPO are combined in the replenishment agent. The approach of not considering forecasting and ordering as a joint problem gives some additional possibilities, since the advantages of different algorithms can be combined. However, none of the mentioned papers with split forecasting and ordering have made a comparison with RL agent with joint consideration or presented their data.

5.2.4 Multiple products

RL is most commonly applied for one item, as presented in Table 5. However, articles that include a larger number of items often achieve good results (Meisheri et al. 2022, Tian et al. 2024, Saha & Rathore 2024). Of the papers regarding multiple items, two different approaches are seen, either single-agent handling multiple items or multi-agent with one item each. With a single-agent, the agent have state variables for all items and therefore, to a greater extent, make decisions considering all items simultaneously. In multi-agent environments, each agent constitutes part of the environment observed by the other agents. This limits the possibility of making joint replenishment decisions. Meisheri et al. (2022) adopts a multi-agent approach with one agent per item. The authors highlight the scalability and parallelization of their approach. Scalability mainly refers to the ease of expanding to a large or variable number of items since this approach "splits the original problem into constant-scale sub-problems". However, Meisheri et al. (2022) does not mention demand dependencies between the items.

Tian et al. (2024) base their solution on a single agent handling multiple items each. Saha & Rathore (2024) applies a multi-echelon approach where each installation is represented by an agent, each agent still handles all items that are processed in each installation. The multi-echelon network consists of multiple Point Of Care (POC) and one Hospital Pharmacy (HP). The POC:s service customers directly and are downstream from the HP. The HP order from multiple pharmaceutical suppliers. Both articles mention the effect of demand correlation on the RL agent(s) and Saha & Rathore (2024) especially highlights this as an advantage enabling more efficient inventory management.

One difference between the articles is the number of items, where Saha & Rathore (2024) test their model on 3 350 items and Tian et al. (2024) on 4 items. Saha & Rathore (2024) is therefore the only article considering a sufficiently large number of items to see the effects of the unexpected demand correlation. This effect is highlighted as a significant improvement with a cost decrease of around 30% compared to an (s, S) policy which disregards both joint replenishment and demand dependencies. The (s, S) policy parameters are determined by hospital inventory managers and hospital pharmacists based on experience and annual consumption. This is highlighted by Saha & Rathore (2024) as a "primary indicator of the inefficiency".

Both Tian et al. (2024), Saha & Rathore (2024) use real data. Saha & Rathore (2024) used a data set over 12 months with 28 000 unique medication orders. The article also covers tests where items are divided by low or high demand and low or high variability. The exact definition of high or low is not disclosed in the article, and the data used is confidential and not published. The demand pattern of "low-demand" is unknown and thus, limited comparisons with spare parts demand can be made.

5.2.5 Constraints in inventory

Depending on the focus area of inventory management, various constraints have varying relevance and importance. As presented in Table 5, the majority of articles do not consider capacity constraints in storage and the most common approach is to not consider a large number of, or extensive, constraints (Bergsma et al. 2025). The more specialized articles on, for example, aerospace and medicine, have constraints such as size, shelf life, and total warehouse limitations (Temizöz et al. 2025, Wang et al. 2022, Burtea & Tsay 2024, Meisheri et al. 2022). These limitations are to a high extent dependent on the specific settings and make inventory optimization more complex. The transferability of RL agents between domains varies depending on constraints. Storage capacity is more relevant for larger products and shelf life has a high applicability on medicine while being a less limiting constraint for spare parts Bergsma et al. (2025), Wang et al. (2022). Another aspect raised by Burtea & Tsay (2024) is the safe deployment of RL in real world settings. Safe deployment focuses on achieving desirable results during actual implementation using realistic constraints. The authors give examples such as avoidance of infeasible inventory solutions that exceed storage capacity or the avoidance of the bullwhip effect.

5.3 Special cases - Disruptions and Transshipments

As mentioned by several research papers, for RL to perform well and as intended, the data used for training should be similar to the test data or future data (Sutton et al. 2018). The challenges with assuming stationary environments are limiting RL usage, therefore addressing and adapting to non-stationary environments is relevant. Padakandla (2021) states three main challenges with non-stationary environments; sample efficiency, computational power, and theoretical results. Sample efficiency refers to efficient use of data in training. When samples do not clearly reflect changes in the underlying process, the algorithm requires more time to learn a new policy. Furthermore, detecting changes in the environment often requires an expanded state space, which increases the computational power needed. For non-stationary environments, assumptions of convergence in policy iteration no longer hold. This limits the performance guaranties for RL algorithms.

Padakandla (2021) states that research on non-stationary environments is scarce and the area requires further research. The conducted research is often focused

on a niche area such as robotics, digital marketing, recommender systems, transportation, etc. Therefore, non-stationary environments remain a challenge for RL algorithms, although this issue is not unique to RL.

A common legal action in systems with multiple retailers is transshipments between retailers. This action often results in a decreased total cost after an ability to, satisfy demand by avoiding stock out (Kim et al. 2024). An alternative usage of RL in inventory management, especially relevant for products with a high requirement on service levels, such as spare parts, is transshipments between independent retailers. Such an arrangement between retailers is most relevant when information sharing is limited and retailers want to retain ordering decision autonomy. This implementation of RL and transshipments was found to increase total profit and decrease unmet demand. A main advantage of this implementation is increased resilience for long- and short-term supply chain disruptions. Kim et al. (2024) also states that increased information sharing, for example customer demand, increased the simulated total profit and could create a more competitive SC. However, a retailer might not be willing to share this information to avoid misuse and create unfair advantages for competitors. In conclusion, RL usage for transshipments in inventory management is a relevant and advantageous method to increase profit and resilience for supply chains when centralized ordering is not applicable.

6 Proposed model

The proposed model is a multi agent Dueling Double Deep Q-Network using Experience buffer Modification and Prioritized experience replay, in short, EM-P3DQN. It is based on the combination of a double and a dueling DQN model proposed by Wang et al. (2016) with the addition of PER, introduced by Schaul et al. (2015). This combination has not been tested in previous research. The Experience buffer modification is used to initialize the experience buffer with an (R,Q)-policy as a teacher policy. Furthermore, the experience buffer utilizes prioritized experience replay. The agent is a dueling double deep Q-network, with both an evaluation and a target network (double), and for both networks including a value and an advantage branch (dueling).

A Q-network was used as the basis of the model, both because this continued the work by Syncron and because the literature review showed more efficient data usage for off-policy methods. The other well-studied alternative, PPO, is an on-policy RL method and is thus less sample efficient. Q-learning allows using experiences multiple times and therefore will extract more information per sample.

Since the model is based on Q-learning, the update of the value of each state and action combination in the MDP is approximated according to Equation 8 presented in Section 3.4. Instead of the target value Y_t used in Equation 8, the target value, $Y_t^{DoubleDQN}$ defined in Equation 9 is applied, effectively utilizing the Q-value from both networks for the target value estimation. The Q-values in question are defined according to Equation 10, exploiting the separation of the value and advantage of a state and action combination.

6.1 State and action space

The agent's state space consists of the inventory position and inventory level for the given item. Other combinations of state space were tested which included a "past demand flag" (including information of the demand in the past days) and "delivery flag" (information on incoming orders for the upcoming lead time). Training with extended state space did not improve performance and decreased convergence for this network.

The action space varied depending on the item and consisted of multiples of the order quantity calculated for the (R,Q)-policy. To limit the action space to not perform unnecessary calculations, the action space included multiples of Q until it covered the largest single instance of demand in the data. The (R,Q)-policy does not need to order multiples of Q outside this range and the action space is therefore not limiting for the (R,Q)-policy, for the RL-policy with less transparency, the same cannot be definitively concluded. Nonetheless, the limitation is assumed to not limit the RL-policy to a great extent.

6.2 Training and Testing data set

Real demand data from 100 different spare parts were used for training and testing. Syncron categorizes the item demand according to the demand characteristics, using the four categories *fast, erratic, lumpy*, and *slow*. Each category except *slow* featured 20 items, while *slow* featured 40 items. 18 months of item data was used for the categorization of demand. The categorization was done in two steps following the flowchart in Figure 4. First, the preprocessed dataset was evaluated on the number of months with zero demands. If an item featured zero demands in more than 62% of the months in the dataset, it was categorized as an item with intermittent demand. As seen in Figure 4, if the demand variance, $\hat{\sigma^2}$, divided by the mean demand, $\hat{\mu}$, was greater than 2.4 the item was categorized as an item with high variation. The value of 2.4 is determined by the case company providing the data and is a chosen value determining the division between low and high variance.

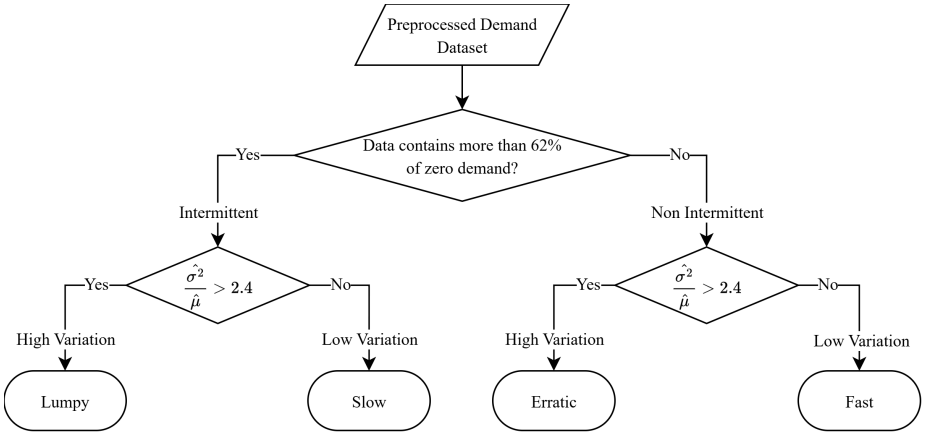

Figure 4: Flowchart describing the categorization of demand types at Syncron

The historical demand is over three years (2023, 2024 and 2025) where the first two years were used for training and the final year was used as validation. Furthermore, each item had information on non-stochastic lead time and a target service level. For all items with *slow* and *lumpy* demand, service levels were specified with S1 and for items with *fast* or *erratic* demand, S2 were used.

Additionally, the data contained Number Of Orders Per Year (NOOPY). NOOPY is the target number of orders per year, predetermined by Syncron and the case company providing the data.

The real demand data was assumed to only include independent items without correlations. In reality, depending on the supply chain setup, correlations may be seen between spare parts and could therefore affect the results. Since all

items were handled independently in both the RL and the (R,Q)-model, no test for dependencies were done.

6.3 Cost evaluation

To evaluate and compare the RL-policies and (R,Q)-policies, a cost approach was used. Since no holding or backorder cost was specified in the data, the costs were calculated based on the target service levels. The cost was based on Equation 2 which gives a relationship between holding cost and backorder cost. Since the absolute cost is less relevant than the relative cost, the holding cost was set to $h = 1$ for all items and the backorder cost was derived from Equation 16. Although the equality in Equation 16 does not strictly hold for the used demand data, it is an assumption that keeps comparability between RL-policy and (R,Q)-policy and is therefore used.

$$b = \frac{S_2}{1 - S_2} \tag{16}$$

For items with a specified target cycle service level instead of item fill rate, an expected item fill rate was used to calculate the backorder cost. This expected item fill rate was obtained by Syncron's previous inventory optimization based on the cycle service level.

6.4 Determining (R,Q) parameters

The (R,Q)-model and the RL agent were confined to ordering multiples of the order quantity Q. As Q was not specified in the data, it was determined for each item using Equation 17.

$$\lceil \frac{\sum_{t=1}^{T} O_t}{Y \cdot NOOPY} \rceil = Q. \tag{17}$$

The total number of demand orders across the full time period T in the training data was divided by NOOPY multiplied by the number of years Y in the data. In this model, $Y = 2$. The acquired value was then rounded up to an integer value Q.

To determine R for each item, different values of R were iteratively tested directly on the training data. For the fixed Q, the total cost (holding and backorder cost) was calculated for each of the R values and the R resulting in the lowest cost was chosen. The range of tested R was chosen as $R = -1$ up until R where all demand is met from stock. Testing for $R < -1$ has theoretical relevance down to $R = -Q$, however, for practical settings, suppliers often require non negative reorder points to satisfy customers. Regarding the upper bound, a higher R than when all demand is met from stock will only increase the inventory, without decreasing backorders and therefore increasing cost. With the chosen R and Q, the (R,Q)-policy was thereafter tested on the test data.

Using a cost approach rather than optimizing R directly based on service level has some limiting implications. For example, if Equation 16 does not accurately represent the relationship between service level and backorder cost, the achieved service level will not meet the target service level. With the used assumptions and simplifications, there is a risk of undershooting both the backorder cost and the achieved service level. Additionally, with limited data and low demand, the achieved service level further poses the risk of undershooting target service level. Therefore this method has more limited applicability to reality than optimizing R based on service level. However, the advantage is that this increases comparability with the RL-policy. Since the primary comparison between the RL-policy and the (R,Q)-policy is made based on costs, it is deemed relevant to make all optimizations based on the same parameter.

6.5 Pretraining

To speed up model training and convergence, pretraining was applied. The pretraining was used to initialize the networks closer to what is believed to give satisfactory results. This is done by training the agent on data and then using the resulting RL network and ordering policy to start training for succeeding item trainings. To adapt the pretrained network to the four different item demand categories, four different pretrained networks were used. For each demand category, one item was chosen to be used for pretraining. The choice of item was based on the characteristics of the item, such as lead time, total demand, and target service level.

6.6 Early stopping

During training, a maximum of 10 000 training episodes was allowed per item. Each episode is a complete run over each day in the training data set for the given item. Additionally, each step included a bootstrap of 256 past experiences which contributed to the resulting network update. Early stopping was used to reduce the extent to which the choice of training episodes affected the results and to minimize the training time by limiting the number of episodes that actually run. The early stopping was implemented using a patience parameter which ensured that the number of training episodes were dynamically dependent on the training result. The patience parameter was set to 2000 episodes and hence, early stopping was applied if the agent did not reach a better training result in 2000 episodes, counted from the last best training result. Furthermore, training results usually varies more in the initial episodes and therefore the early stopping was set to start after 1000 episodes. This was implemented to limit the impact from the pretrained network and let the new network adapt to the specific environment.

6.7 Experience buffer modification and PER

Since data shortage is a general problem for spare parts, methods to use the data intelligently are key. Therefore, both experience buffer modification and PER was implemented in the model. The teacher policy used for experience buffer modification was the (R,Q)-policy determined according to Section 6.4 which was used to prefill the experience buffer with 4000 teacher experiences. These experiences are transitions between two states where the agent acts as the (R,Q)-policy would. As old experiences with low priority (low TD error) are removed from the experience buffer to limit the total number of experiences, a new teacher experience was generated every 100 episodes to ensure that the experience buffer always contains teacher experiences.

PER was used to leverage and prioritize transitions with unexpected discrepancies between expected reward (Q-values) and received reward. By prioritizing the use of experiences, the data was used more efficiently in training.

7 Results

The results are primarily measures on two metrics, daily cost and cost ratio, taken as averages over all items in the study. The daily cost per item is calculated as the total cost over the entire time horizon of N days for a given item, see Equation 18. The daily cost per item results in one value each for RL- and (R,Q)-policy for each item. The cost ratio per item is the ratio between the daily costs for the (R,Q)-policy and the RL-policy, see Equation 19.

$$\frac{\sum_{n=1}^{N} \text{Cost}_n}{N} = \text{Daily Cost per item} \tag{18}$$

$$\frac{\text{Daily Cost}_{(R,Q)}}{\text{Daily Cost}_{RL}} = \text{Cost ratio per item} \tag{19}$$

7.1 Initial Analysis and Outliers

To confidently draw conclusions from the aggregated results, 7 items were excluded due to lack of data. Three of these items had zero demand on test data and four items had zero demand on training data. Performance on items without training data is mostly dependent on initialization of the models, and it is not possible to confidently evaluate performance on items without test data.

Furthermore, a total of 15 items were identified as outliers, 12 with high cost differences between RL and (R,Q)-policies, and 3 with high cost difference between training and test data. Of the 3 items with difference between training and test, 2 varied for the (R,Q)-policy and 1 varied for the RL-policy. The training and test cost is presented in Table 8. The high test cost was caused by high backorder cost following high target service levels of 0.9999 for all three items and a policy from training data resulting in high backorders on test data. These high discrepancies suggest inconsistent training and test data.

Table 8: Daily cost per item on three outlier items. Item 1 and 2 - results from the (R,Q)-policy and item 3 - result from RL-policy

	Item 1	Item 2	Item 3
TSL	0,9999	0,9999	0,9999
Training cost	8,1	9,8	2,5
Test cost	4800	666	363

12 items were defined as outliers where the RL result differed by more than a factor of 10 compared to the (R,Q)-policy. 11 of these had an RL cost higher than that from the (R,Q)-policy. These items skewed the overall averages and will therefore be handled partly separately. Although no definitive explanation could be identified, non-convergence and overtraining are two plausible causes. Of these, non-convergence appears to be the more likely explanation, given both

the limited extent of hyperparameter tuning and the inherent convergence challenges associated with DQN. Furthermore, the risk of overtraining was partially mitigated through the use of early stopping.

7.2 Averages across items

Focusing first on the average daily cost taken over all items in the study, Table 9 shows that the proposed RL-policy does not outperform the considered (R,Q)-policy. For the average daily cost over the test data presented in Table 9, the (R,Q)-policies achieved an average daily cost of 85.970 while the RL-policies achieved an average daily cost of 185.871, 118% higher. The corresponding measurement of the result when excluding the 15 outliers is presented in Table 10. There we see that the RL-policies reach an average daily cost of 15.919, 95% higher than the (R,Q)-policies. Although the RL-policy incurs approximately 100% higher average daily costs than the (R,Q)-policy both with and without outliers, the absolute cost levels differ substantially. When outliers are included, the average daily cost is more than ten times higher than when excluding outliers.

Table 9: Performance comparison RL and (R,Q) across items

	training		test	
Metric	RL	(R,Q)	RL	(R,Q)
avg daily cost	65.539	5.969	185.871	85.970
avg cost ratio	1.217		1.098	
σ of avg cost ratio	1.788		1.256	
avg backorders	0.268	0.152	0.357	0.218
avg inventory	8.804	4.250	9.164	4.178
share of items fulfilling TSL	0.096	0.223	0.170	0.383
share of RL better than (R,Q)	0.287		0.245	

Table 10: Performance comparison without outliers across items

	Training		Test	
Metric	RL	(R,Q)	RL	(R,Q)
avg daily cost	13.942	6.220	15.919	8.162
avg cost ratio	1.085		1.018	
σ of avg cost ratio	1.020		1.112	
avg backorders	0.286	0.165	0.338	0.189
avg inventory	5.897	4.319	5.976	4.354
share of items fulfilling TSL	0.077	0.179	0.128	0.397
share of RL better than (R,Q)	0.295		0.282	

Turning to the average cost ratio taken over the items in the study, we first

note that a low value indicates low RL performance and a value over 1 indicates the RL-policy outperforming the (R,Q)-policy. Tables 9 and 10 show that the RL-model on average outperforms the (R,Q)-policies to a slight degree, with an average cost ratio of 1.018 on test data. The contradiction between *average daily cost* and *average cost ratio* follows from poor performance of RL on costly fast moving items increasing the averages, with better performance on slow moving items. Fast moving items with high target service levels result in higher daily costs per item for handling the inventory. The range between the most costly and the cheapest items is presented in Table 11. Furthermore, outliers has a higher impact on the average daily cost than the average cost ratio. Since the cost ratio per item does not consider the absolute cost, the cost ratio per item has a lower range between highest and lowest value and only compares relative performance between the policies.

Table 11: Highest and lowest daily cost for both RL and (R,Q) and cost ratio per item for the tested items.

	Daily cost		Cost ratio per item
Metric	RL	(R,Q)	
Highest	9906	4801	13.88
Lowest	0.005684	0.005668	0.002205

To illustrate how daily cost and cost ratio per item are affected by high and low demand, two items are compared. The first item resulted in an absolute cost difference between (R,Q) and RL of 882, and a cost ratio of 0.86, corresponding to 14% lower costs for (R,Q). The second item resulted in an absolute cost difference of 0.52 and a cost ratio of 5.6, corresponding to 460% higher cost for (R,Q). Although the second item had a far higher percentage deviation, the absolute daily cost difference is lower. This follows since the first item had higher demand and higher target service level, while the second item had relatively low demand and low target service level. Consequently, an identical absolute increase in daily cost for the RL-policy compared to the (R,Q)-policy can result in substantially different cost ratios across items. Thus may relatively small absolute cost differences translate into large cost ratio deviations for low cost items, whereas much larger absolute differences may correspond to only modest cost ratio deviations for high cost items.

Moreover, the standard deviation presented in Tables 9 and 10 of the average cost ratio is relatively high for both training and test results. This indicates low confidence in the results and a considerable risk that the obtained results are influenced by randomness. The high variability and randomness observed in the results indicate a lack of consistent performance across items. This behavior is likely a consequence of convergence issues in the RL-model. However, due to the black-box nature of reinforcement learning, it is difficult to determine the underlying causes of this behavior. Furthermore, the comparison of the average daily cost is somewhat skewed by the assumption of a common holding cost

of $h = 1$ for all items. This assumption disregards different item values and, therefore, makes comparisons between items less accurate.

Other relevant metrics to compare are the *service level*, *backorders*, and *inventory* independently. Based on Table 10 it can be noted that the (R,Q)-policy performed better on these metrics. This applies for data both with and without outliers.

Further potential is shown where the RL-policy outperforms the (R,Q)-policy on cost in just under 30% of items. Additionally, 69% of all RL-policies reached less than 20% cost increase compared to an (R,Q)-policy. For items with training RL performance better than (R,Q) performance, there is a probability of around 50% to outperform the (R,Q)-policy on the test data.

Furthermore, training of the RL agent often showed volatility in training results over episodes. This resulted in slightly different performance of the RL-policy for different trainings on the same item. Although the patience mentioned in Section 6.6 aimed to mitigate this, it highlights the complexity of convergence in RL-models.

7.3 Different types of demand

The performance of the RL-model on different demand types varies. The results divided for each demand type are presented in Appendix 10. For average daily cost on training and test data, the RL-policy did not consistently outperform an (R,Q)-policy in any of the four demand types. Regarding the average cost ratio, it is noted that the RL-policy outperformed the (R,Q)-policy on test data for slow and lumpy demand while the (R,Q)-policy performed better on fast and erratic demand. On the training data, results were the opposite for erratic and slow. However, metrics are in general less consistent and the differences between (R,Q) and RL, and training and testing, are greater within the divided demand types. This is likely partly due to the decrease in sample size. Due to this, conclusions should be made with caution.

7.4 Categorizing items by service level and lead time

By categorizing the items by target service level and lead time, some patterns may be noticed. In average daily costs, the (R,Q)-policy consistently outperforms the RL-model for all tested categories. When instead comparing the average cost ratio, the (R,Q)-model tends to perform better on low target service levels, whereas the (R,Q)-policy has better performance over higher service levels. The service levels were categorized in three categories, $SL \leq 0.9$, $0,9 < SL \leq 0.95$ and $0.95 < SL$. In the first category, the resulting average cost ratio was 1.26 while it was 1.07 for the last category. Regarding lead time,

a similar pattern was identified. The average cost ratio was 1.25 for the shortest lead times. The lead time was divided into the following three categories; $LT \leq 12$, $12 < LT \leq 60$ and $LT > 60$. For the two last categories of lead time, the average cost ratio was 0.65. It should also be noted that the categories does not contain the same number of items.

The exact cause of the competitive performance on short lead times and lower service levels has not been determined. It should also be mentioned that low service levels, short lead times, and the four demand types are correlated. Therefore, further analysis and research is needed.

7.5 Achieved Service Levels

7.5.1 Compared to TSL

A comparison was made between the achieved service level and TSL for both RL and (R,Q) for all demand types. As can be seen in Table 12 both the RL and (R,Q)-policies resulted in a service level undershoot. The RL-policies exhibited a larger average undershoot of the target service level than the (R,Q)-policies. In contrast, the (R,Q)-policies exhibited a lower standard deviation than the RL-policies for all demand type categories except *Fast*. The RL-policies had an average comparable performance to the (R,Q)-policies for items belonging to demand type *Slow* and *Lumpy*. This is consistent with the histogram distributions seen in Section 7.5.2.

Table 12: Service level undershoot on test data compared to TSL

	RL	σ_{RL}	**(R,Q)**	$\sigma_{(R,Q)}$
Average tot	0.320	0.350	0.219	0.396
Fast	0.323	0.251	0.094	0.196
Erratic	0.258	0.172	0.161	0.226
Slow	0.245	0.463	0.180	0.532
Lumpy	0.548	0.225	0.519	0.228

It can be concluded that both RL and (R,Q) does not have a satisfactory result compared to the TSL. As discussed in Section 6.4, the low performance might not only be caused by RL related problems but also by the optimization metric. For that reason, to handle the low achieved service levels, a different reward approach is relevant. For the (R,Q)-policy, this could be done by optimizing directly towards service levels instead of inaccurate holding and backorder costs.

Table 13: Average TSL and backorder cost per demand type

Demand Type	Average TSL	Average Backorder Cost
Overall	0.930	18.147
Fast	0.971	34.082
Erratic	0.976	41.079
Slow	0.899	6.799
Lumpy	0.900	1.975

The different demand types have, as seen in Table 13, different average TSL. This in turn impacts the backorder cost which differs with a factor of over 20 between demand type *Erratic* and *Lumpy*. It is important to note that the backorder cost for *Slow* and *Lumpy* is based on the expected item fill rate derived from previously performed inventory optimization by Syncron AB, see Section 6.3. The difference in backorder cost largely impacts the ability for both the RL-policies and (R,Q)-policies to achieve higher service levels. This is apparent when looking at the undershoot in Table 12 where *Slow* have higher standard deviation and *Lumpy* have bigger undershoot for both RL and (R,Q). It is also apparent when looking at Figures 5-8, where achieved service levels for *Slow* and *Lumpy* are much lower than for *Fast* and *Erratic*.

7.5.2 Achieved service levels for each demand type

In this section the achieved service level for both RL and (R,Q)-policies, for each demand type, are compared. This is done using histograms which sum up service level results for individual items in increments of 10%. The comparisons were made on the test data to portray the real performance on TSL for each policy. In Table 14 the average service level per demand and policy type are presented.

Table 14: The average achieved service levels per demand type and policy

	RL avg SL	(R,Q) avg SL
Fast	0.648	0.877
Erratic	0.718	0.815
Slow	0.573	0.638
Lumpy	0.349	0.379

For items of demand type *Fast*, the achieved service levels can be viewed in Figure 5. In the histogram it can be seen that the service level for the (R,Q)-policy is weighted towards $\sim 100\%$ while the service level for the RL-policy does not exhibit such a pattern. The RL-policy does not exhibit a clear pattern.

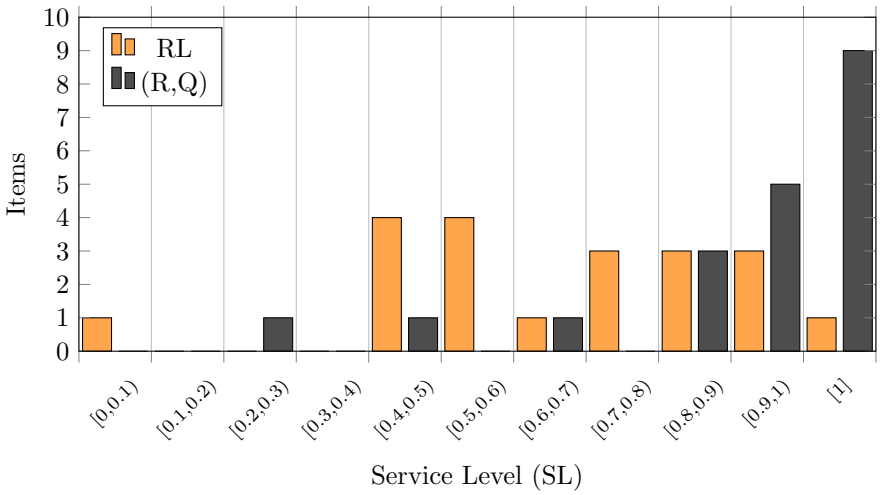

Figure 5: Service Level for Fast Demand type items

For items with demand type *Erratic*, results are visualized in Figure 6. In the histogram, both the RL and (R,Q)-policy exhibit similar patterns, although with shifted means. It can be seen that the (R,Q)-policy has a lower worse case but, for 40% of the items, achieves a service level of 100%.

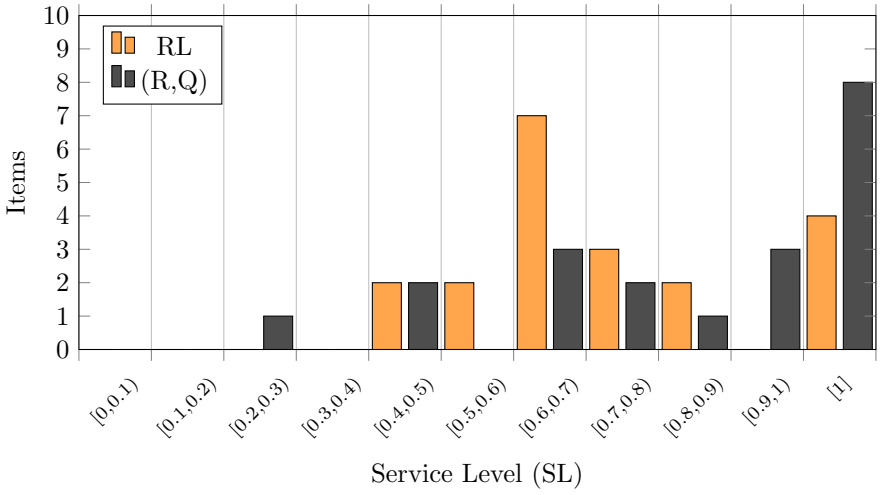

Figure 6: Service Level for Erratic Demand type items

The histogram for items of demand type *Slow*, Figure 7, shows that the distribution of item service levels differs for the RL and the (R,Q)-policy while not differing more than 6.5 percentage points for the mean achieved service level, see Table 14.

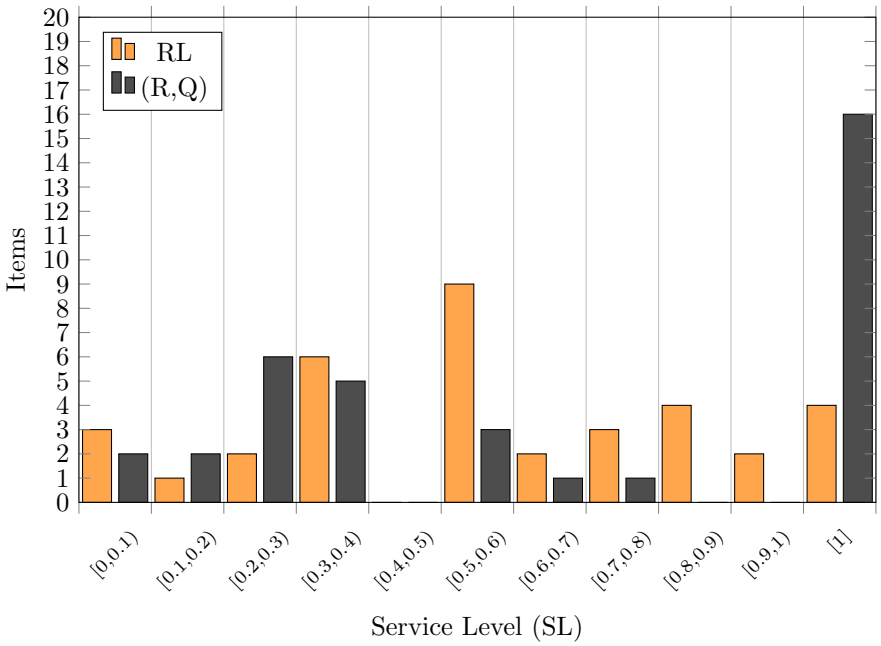

Figure 7: Service Level for Slow Demand type items

In Figure 8, the histogram for items with demand type *Lumpy* is visualized. As seen in Table 14, the average service level for the RL and (R,Q)-policies only differ 3 percentage units. Furthermore, the histogram shows comparable service level performance between the RL- and (R,Q)-policies.

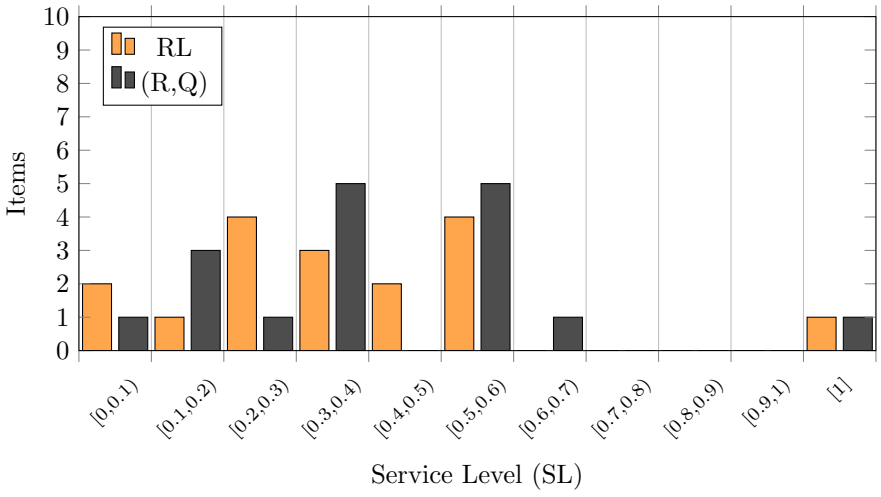

Figure 8: Service Level for Lumpy Demand type items

When comparing Figures 5-8 with each other, it can be seen that items with demand type *Fast* and *Erratic* have overall higher achieved service levels than *Slow* and *Lumpy*. This difference is not only driven by variations in TSL, but could also be a symptom of the cost metric used. For *Slow* and *Lumpy* demand, the low demand rates and frequent zero-demand periods reduce the impact of backorder costs relative to holding costs over time.

7.6 Acceptance for DRL in inventory management

The acceptance for use of DRL in inventory management is a highly relevant issue to consider. Initially, consistent and predictable performance is a likely requirement for customers to accept applications of DRL. It is frequently seen as an area of further research and as seen in the literature review, addressed in recent work.

The black-box characteristics of DRL and the limited understanding of the cause of decisions are other challenges and obstacles to acceptance and implementation. This in combination with convergence and performance issues further pose a hindrance for acceptance.

While acceptance of RL is a challenge, that challenge is also relevant for classical methods of inventory optimization. For a customer representative not specialized within inventory optimization, analytical or heuristic solutions to these problems will likely carry the similar black-box characteristics due to their complexity.

8 Single agent - multi item

Some additional experiments were made with a single agent, multiple item approach. This approach featured an agent capable of handling all or multiple items simultaneously, in order to find a global solution better than the combination of all isolated solutions. Some of the potential advantages of this is the ability to consider demand correlations between items, joint shipments and orders, and consolidate goods to reduce transportation costs and emissions.

8.1 Model Architecture

This approach was based on the model used for the multi agent approach described in Section 6. To handle the increased number of actions, a branching structure between items' actions was adopted. This branching structure is part of the advantage part where the agent chooses one action per item. Using this method, the total number of actions increases linearly instead of exponentially, with increasing number of items. This enables the agent to make individual decisions for each item with a minimal number of output nodes. The tested agent used a network structure where an initial common layer has 128 nodes, followed by a value part with one node and one advantage part per item. The network structure is presented in Figure 9. The reward calculations in the single agent approach used a divided reward between items, instead of a collective reward. This was implemented to directly tie actions related to a specific item result to item reward. This divided reward also aimed to increase convergence in the model compared to a collective reward function.

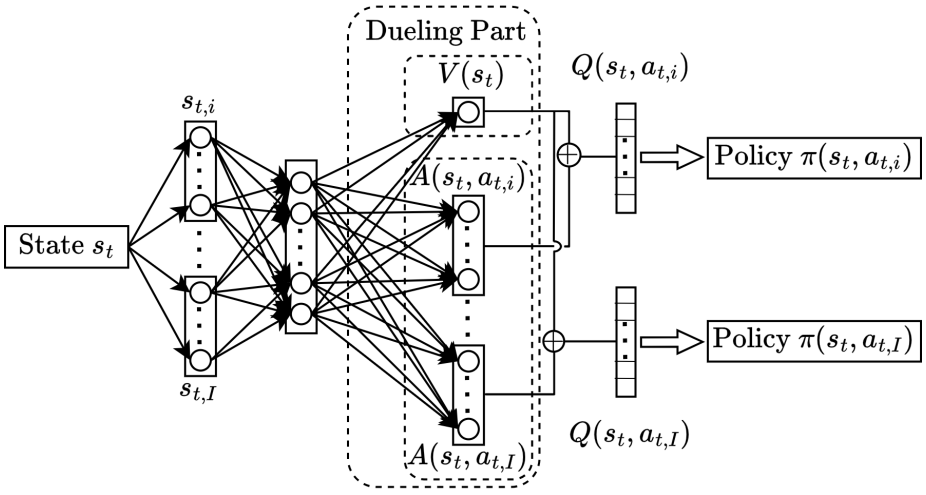

Figure 9: The architecture of the single agent model approach

8.2 Model Performance

The performance of the multi-item single-agent was poor. The added complexity to the model resulted in policies that could not compete with either the (R,Q)-policy or the multi agent approach. The performance decreased when the number of items considered increased. For a higher number of items, the model did not converge. For a limited number of items, the model did converge, but without competitive performance. Some limited tests were conducted on a larger network with both more hidden layers and a higher number of nodes per layer. The larger network structure was motivated by the capability to handle complex dependencies between all 100 items. The added complexity to the model resulted in both additional hyperparameters and higher computational requirements. It can therefore not be excluded that the model may have competitive performance with additional hyperparameter tuning and more training.

The poor model performance and convergence do not, however, ensure that a single agent multiple item approach may not converge for complex problems with competitive results. This approach likely has potential with further hyperparameter tuning and more computational power. For example, Saha & Rathore (2024) showed the possibilities with a single agent, multi-item approach to an inventory management system. Since their model and approach are not publicly available, their exact solution is unclear.

9 Proposal to Syncron

The main proposal to Syncron is to keep testing and be responsive towards the continuously developing research within DRL in inventory management. As of now, DRL is not a suitable solution to all inventory management settings, and to be resource effective, it is essential to identify applicable situations and have focused development.

A category of situations with high potential is when more information than is currently used is accessible. This applies to for example systems with high demand correlation, complex multi-echelon systems and environments with information on predictive maintenance. DRL have been applied in previous research to all these areas with improved performance and cost savings. Therefore, it is recommended to keep these areas in long term focus during the development and testing of DRL.

One main challenge is to handle the lack of convergence. Convergence problems result in poor and unpredictable inventory policies. There are several newly published proposals for this problem, with the most novel solution being the use of estimate of the expected reward. Two other potential solutions are more thorough hyperparameter tuning and using another DRL algorithm than DQN to prioritize stability over efficient data usage. The exact performance of these solutions have not been explored in this thesis, but have shown potential in previous research.

Although complex situations and systems with additional information is seen as a long term goal, handling convergence problems likely enables predictable and competitive performance on simpler systems. This in itself enables partial development, testing, and application to smaller and more controllable environments. Partial development could be a more feasible procedure for DRL development.

In conclusion, DRL shows great potential for implementation in inventory management, but it is important to clarify a goal with the development and relevant environments for DRL usage. Convergence is a main problem for further implementation of DRL with a handful of proposed solutions presented in recently published research, as well as an area of continuous development. Applications with the greatest long term potential are seen as complex environments, but with simpler environments as a potential development.

10 Conclusion

Inventory management of spare parts often involves complex problems divided into smaller subproblems that are solvable for relatively lightweight heuristics. Handling inventory management in a more efficient way and considering additional information have great potential to reduce costs and improve efficiency. Deep reinforcement learning has potential to handle these problems more efficiently than the currently used heuristics.

The main upside of DRL seems to be seen when applied to complex ordering structures or additional information is considered, such as demand correlation or multi-echelon approaches. These applications also come with great complexity and challenges regarding for example, convergence, insufficient data, and unstable environments.

The proposed model does not "consistently outperform (R,Q)-policies for inventory management of single-echelon intermittent spare parts demand" as stated in RQ1. The RL-policies resulted in about double the average cost of inventory while also resulting in around 10% decrease in average cost ratio. The model shows a possible potential for applying RL in simpler settings. Compared to the application of (R,Q)-policies, the RL-model performed the best on low demand, while having less competitive performance on higher demand. However, this result does not ensure that an RL-model may not consistently outperform (R,Q)-policies or other relevant inventory policies. With further network and hyperparameter tuning, as well as potential implementation of additional algorithms, the RL performance will likely increase. For example, if the convergence issues of the proposed RL-model were to be addressed, the RL-model may be able to consistently outperform (R,Q)-policies.

The proposed model was unable to achieve the TSL for any of the demand types in the data set. While the (R,Q)-policies did not achieve the TSL either, they still outperformed the RL-policies. If it is to be implemented in reality, this points to the need for a more SL oriented model or a more accurate cost estimation for example not assuming that all holding costs are equal to one but rather related to the value of the item as well as backorder costs that accurately reflect the costs incurred when missing an order. Furthermore, this may not be a problem with RL but rather a problem with the reward structure of the proposed model.

There are several opportunities for further research within DRL in inventory management. This is especially relevant for more complex situations and testing on real demand data and applications to real situations. Apart from increasing performance, further research will help mitigate the potential acceptance problem if substantial testing has been performed in real settings.

References

Abolghasemi, M., Beh, E., Tarr, G. & Gerlach, R. (2020), 'Demand forecasting in supply chain: The impact of demand volatility in the presence of promotion', *Computers & Industrial Engineering* **142**.

Abu Zwaida, T., Pham, C. & Beauregard, Y. (2021), 'Optimization of inventory management to prevent drug shortages in the hospital supply chain', *Applied Sciences* **11**(6), 2726.

Ahmed, N., Wahed, M. & Thompson, N. C. (2023), 'The growing influence of industry in ai research', *American Association for the Advancement of Science* .

Andersson, H., Hoff, A., Christiansen, M., Hasle, G. & Løkketangen, A. (2010), 'Industrial aspects and literature survey: Combined inventory management and routing', *Computers & Operations Research* **37**(9), 1515–1536.

Andrey Meshalkin, Syncron (2026), 'Why spare parts demand forecasting is a different game', `https://www.syncron.com/blog/why-spare-parts-demand-forecasting-is-different`. Accessed: 24 January 2026.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O. & Zaremba, W. (2017), 'Hindsight experience replay', *Advances in neural information processing systems* **30**.

Axsäter, S. (2015), *Inventory control*, Springer.

Babai, M., Arampatzis, M., Hasni, M., Lolli, F. & Tsadiras, A. (2025), 'On the use of machine learning in supply chain management: a systematic review', *IMA Journal of Management Mathematics* **36**(1), 21–49.

Batsis, A. & Samothrakis, S. (2024), 'Contextual reinforcement learning for supply chain management', *Expert Systems with Applications* **249**, 123541.

Bergsma, R., de Ruijt, C. & Bhulai, S. (2025), 'A systematic review of machine learning approaches in inventory control optimization', *Operations Research Perspectives* p. 100367.

Boute, R. N., Gijsbrechts, J., Van Jaarsveld, W. & Vanvuchelen, N. (2022), 'Deep reinforcement learning for inventory control: A roadmap', *European journal of operational research* **298**(2), 401–412.

Burtea, R. & Tsay, C. (2024), 'Constrained continuous-action reinforcement learning for supply chain inventory management', *Computers & Chemical Engineering* **181**, 108518.

Geevers, K., Van Hezewijk, L. & Mes, M. R. (2024), 'Multi-echelon inventory optimization using deep reinforcement learning', *Central European Journal for Operations Research* **32**(3), 653–683.

Gijsbrechts, J., Boute, R. N., Van Mieghem, J. A. & Zhang, D. J. (2022), 'Can deep reinforcement learning improve inventory management? performance on lost sales, dual-sourcing, and multi-echelon problems', *Manufacturing & Service Operations Management* **24**(3), 1349–1368.

Goltsos, T. E., Syntetos, A. A., Glock, C. H. & Ioannou, G. (2022), 'Inventory–forecasting: Mind the gap', *European journal of operational research* **299**(2), 397–419.

Hillier, F. S. (2005), *Introduction to operations research*, McGrawHill.

Höst, M., Regnell, B. & Runeson, P. (2006), *Att genomföra examensarbete*, Studentlitteratur AB.

Hu, J., Xia, L., Huang, T. & Wu, H. (2025), 'A multi-agent deep reinforcement learning approach for multi-echelon inventory optimization and its application to the beer game', *Transportation Research Part E: Logistics and Transportation Review* **203**, 104367.

Ilie, M. & Semenescu, A. (2025), 'Monte carlo markov chain (mcmc) stochastic modeling of supply chain', *Series on engineering sciences (Academy of Romanian Scientists)* .

Johansson, L. (2026), 'Principal data scientist at syncron', *Interview* .

Kim, B., Kim, J. G. & Lee, S. (2024), 'A multi-agent reinforcement learning model for inventory transshipments under supply chain disruption', *IISE Transactions* **56**(7), 715–728.

Kirk, R., Zhang, A., Grefenstette, E. & Rocktäschel, T. (2023), 'A survey of zero-shot generalisation in deep reinforcement learning', *Journal of Artificial Intelligence Research* **76**, 201 – 264.

Lengu, D., Syntetos, A. A. & Babai, M. Z. (2014), 'Spare parts management: Linking distributional assumptions to demand classification', *European Journal of Operational Research* **235**(3), 624–635.

Li, K., Rubungo, A. N., Lei, X., Persaud, D., Choudhary, K., DeCost, B., Dieng, A. B. & Hattrick-Simpers, J. (2025), 'Probing out-of-distribution generalization in machine learning for materials', *Communications Materials* **6**(1), 9.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N. M. O., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. P. (2020), 'Continuous control with deep reinforcement learning'. US Patent 10,776,692.

Liu, X., Hu, M., Peng, Y. & Yang, Y. (2025), 'Multi-agent deep reinforcement learning for multi-echelon inventory management', *Production and Operations Management* **34**(7), 1836–1856.

Meisheri, H., Sultana, N. N., Baranwal, M., Baniwal, V., Nath, S., Verma, S., Ravindran, B. & Khadilkar, H. (2022), 'Scalable multi-product inventory control with lead time constraints using reinforcement learning', *Neural Computing and Applications* **34**(3), 1735–1757.

Nahmias, S. (1994), 'Demand estimation in lost sales inventory systems', *Naval Research Logistics (NRL)* **41**(6), 739–757.

Ohlsson, M., Edén, P. & Brebion, V. (2025), 'Introduction to artificial neural networks and deep learning', *Lund University Course Material for EXTQ41* .

Oroojlooyjadid, A., Nazari, M., Snyder, L. V. & Takáč, M. (2022), 'A deep q-network for the beer game: Deep reinforcement learning for inventory optimization', *Manufacturing & Service Operations Management* **24**(1), 285–304.

Padakandla, S. (2021), 'A survey of reinforcement learning algorithms for dynamically varying environments', *ACM Computing Surveys (CSUR)* **54**(6), 1–25.

Pinçe, Ç., Turrini, L. & Meissner, J. (2021), 'Intermittent demand forecasting for spare parts: A critical review', *Omega* **105**, 102513.

Roy, A., Xu, H. & Pokutta, S. (2017), 'Reinforcement learning under model mismatch', *Advances in neural information processing systems* **30**.

Rozhkov, M., Alyamovskaya, N. & Zakhodiakin, G. (2025), 'The beer game bullwhip effect mitigation: a deep reinforcement learning approach', *International Journal of Production Research* **63**(18), 6630–6647.

Runeson, P. & Höst, M. (2009), 'Guidelines for conducting and reporting case study research in software engineering', *Empirical software engineering* **14**(2), 131–164.

Saha, E. & Rathore, P. (2024), 'A smart inventory management system with medication demand dependencies in a hospital supply chain: A multi-agent reinforcement learning approach', *Computers & industrial engineering* **191**, 110165.

Samuel, A. L. (1959), 'Some studies in machine learning using the game of checkers', *IBM Journal of research and development* **3**(3), 210–229.

Schaul, T., Quan, J., Antonoglou, I. & Silver, D. (2015), 'Prioritized experience replay', *arXiv preprint arXiv:1511.05952* .

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

Smith, B., Garg, A. & Rich, W. (2025), 'Inventory management using reinforcement learning', *South African Journal of Industrial Engineering* **36**(1), 104–118.

Stranieri, F., Stella, F. & Kouki, C. (2024), 'Performance of deep reinforcement learning algorithms in two-echelon inventory control systems', *International Journal of Production Research* **62**(17), 6211–6226.

Sutton, R. S., Barto, A. G. et al. (2018), *Reinforcement learning: An introduction*, Vol. 1, MIT press Cambridge.

Temizöz, T., Imdahl, C., Dijkman, R., Lamghari-Idrissi, D. & van Jaarsveld, W. (2025), 'Deep controlled learning for inventory control', *European Journal of Operational Research* **324**(1), 104–117.

Tian, R., Lu, M., Wang, H., Wang, B. & Tang, Q. (2024), 'Iacppo: A deep reinforcement learning-based model for warehouse inventory replenishment', *Computers & Industrial Engineering* **187**, 109829.

Torrey, L. & Taylor, M. (2013), Teaching on a budget: Agents advising agents in reinforcement learning, *in* 'Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems', pp. 1053–1060.

van der Haar, J. F., van Jaarsveld, W. L., Basten, R. J. & Boute, R. N. (2026), 'Industrializing deep reinforcement learning for operational spare parts inventory management', *European Journal of Operational Research* .

Van Hasselt, H., Guez, A. & Silver, D. (2016), Deep reinforcement learning with double q-learning, *in* 'Proceedings of the AAAI conference on artificial intelligence', Vol. 30.

Wang, G., Li, Y., Cheng, C. X., Li, R., Ding, C. X., Zhang, Y. & Liu, Y. (2025), 'Integrated optimization of equipment degradation modeling and spare parts inventory for predictive maintenance in power systems', *Sustainable Energy Technologies and Assessments* **83**, 104626.

Wang, H., Tao, J., Peng, T., Brintrup, A., Kosasih, E. E., Lu, Y., Tang, R. & Hu, L. (2022), 'Dynamic inventory replenishment strategy for aerospace manufacturing supply chain: combining reinforcement learning and multi-agent simulation', *International Journal of Production Research* **60**(13), 4117–4136.

Wang, S. & Li, F. (2025), 'Deep reinforcement learning with convolutional neural networks for optimizing supply chain inventory management', *Informatica* **49**(26).

Wang, W. (2025), *Principles of Machine Learning - three perspectives*, Springer.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M. & Freitas, N. (2016), Dueling network architectures for deep reinforcement learning, *in* 'International conference on machine learning', PMLR, pp. 1995–2003.

Watkins, C. (1989), JCH (1989). Learning from delayed rewards, PhD thesis, PhD Thesis, University of Cambridge, England.

Yan, Y., Chen, F. Y., Fu, Z. & Bi, W. (2026), 'Heuristics and deep reinforcement learning for the inventory problem with an all-or-nothing yield pattern and non-zero leadtimes', *Computers & Operations Research* .

Yin, R. K. (2018), *Case Study Research and Applications: Design and Methods*, Sage.

Yu, C., Zhou, Y. & Zhang, Z. (2020), Multi-agent reinforcement learning for dynamic spare parts inventory control, *in* '2020 Global Reliability and Prognostics and Health Management (PHM-Shanghai)', IEEE, pp. 1–6.

Yunxiang, G. & Zhao, W. (2024), 'Research on supply chain optimization and management based on deep reinforcement learning', *Scalable Computing: Practice and Experience* **25**(6), 4814–4824.

Zheng, M., Su, Z., Wang, D. & Pan, E. (2024), 'Joint maintenance and spare part ordering from multiple suppliers for multicomponent systems using a deep reinforcement learning algorithm', *Reliability Engineering & System Safety* **241**, 109628.

Zhou, X., Feng, L., Zhu, A. & Shi, H. (2026), 'Uncertainty-aware joint inventory-transportation decisions in supply chain: A diffusion model-based multi-agent reinforcement learning approach with lead times estimation', *Computers & Chemical Engineering* p. 109567.

Zhou, Y., Guo, K., Yu, C. & Zhang, Z. (2024), 'Optimization of multi-echelon spare parts inventory systems using multi-agent deep reinforcement learning', *Applied Mathematical Modelling* **125**, 827–844.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H. & He, Q. (2021), 'A comprehensive survey on transfer learning', *Proceedings of the IEEE* **109**(1), 43–76.

Appendix I Initial RL-model

Table 15: Demand pattern for initial RL-model

Day	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**	**Sunday**
Demand	0	2	0	3	4	0	0

Table 16: Inventory comparison between (R,Q) policy and Q-learning

		IP and (stock on hand)		Order sizes		
Day	Demand	RQ policy	Q-Learning	RQ policy	Q-Learning	Stock difference
Sat		4 (2)	2 (0)	2	0	-2
Sun		6 (4)	2 (2)	2	2	-2
Mon		8 (6)	4 (2)	2	2	-2
Tue	2	6 (6)	4 (2)	2	2	-4
Wed		8 (6)	6 (2)	0	2	-4
Thu	3	5 (5)	5 (3)	2	2	-2
Fri	4	3 (1)	3 (1)	2	2	
Sat		5 (3)	5 (3)	2	2	
Sun		7 (5)	7 (5)	0	0	
Mon		7 (7)	7 (7)	0	0	
Tue	2	5 (5)	5 (5)	2	2	
Wed		7 (5)	7 (5)	0	0	
Thu	3	4 (4)	4 (4)	2	0	
Fri	4	2 (0)	0 (0)	2	2	

Appendix II Hyperparameters

Table 17: Hyperparameters used for the Single Item Double Dueling Deep Q-Network (DDDQN) with experience modification buffer

Parameter	**Value**
Learning and Exploration	
Learning rate (α)	1×10^{-5}
Discount factor (γ)	0.99
Exploration (ϵ)	0.01
Target network update frequency	1000
Neural Network Architecture (identical for double network)	
Common hidden layer	32
Value stream layer	1
Advantage stream layer	$\mathcal{A}$
Training	
Batch size	64
Replay buffer size	100 000
Max number of episodes	10 000
Prioritized Experience Replay	
Prioritizing coefficient (α)	0.6
Bias correction (β)	0.4
Beta increment ($\beta_{\text{increment}}$)	1×10^{-5}

Appendix III Results from different demand types

Table 18: Performance comparison for fast demand

	Training		Test	
	RL	(R,Q)	RL	(R,Q)
avg daily cost	10.187	7.445	9.994	6.109
avg relative cost	0.737		0.852	
avg backorders	0.155	0.058	0.160	0.019
avg inventory	4.379	5.111	4.257	5.161
share of items fulfilling SL	0.063	0.313	0.063	0.5
share of RL better than (R,Q)	0		0.313	

Table 19: Performance comparison for erratic demand

	Training		Test	
	RL	(R,Q)	RL	(R,Q)
avg daily cost	43.633	15.321	53.682	26.097
avg relative cost	1.410		0.745	
avg backorders	0.431	0.120	0.548	0.308
avg inventory	18.560	12.368	20.598	12.007
share of items fulfilling SL	0.176	0.353	0.176	0.412
share of RL better than (R,Q)	0.412		0.176	

Table 20: Performance comparison for slow demand

	Training		Test	
	RL	(R,Q)	RL	(R,Q)
avg daily cost	5.163	2.592	3.950	2.179
avg relative cost	0.818		1.357	
avg backorders	0.237	0.144	0.234	0.091
avg inventory	2.269	1.332	0.815	1.581
share of items fulfilling SL	0.069	0.1	0.167	0.5
share of RL better than (R,Q)	0.268		0.4	

Table 21: Performance comparison for lumpy demand

	Training		Test	
	RL	(R,Q)	RL	(R,Q)
avg daily cost	4.393	1.856	4.355	1.992
avg relative cost	1.199		1.137	
avg backorders	0.288	0.373	0.362	0.432
avg inventory	1.934	0.329	1.274	0.363
share of items fulfilling SL	0.085	0	0.194	0.087
share of RL better than (R,Q)	0.452		0.290	