The adoption of reinforcement learning in the logistics industry: A case study at a large international retailer

MSc Business Information Technology

M.W.T. Gemmink

UNIVERSITY OF TWENTE.

Master thesis

The adoption of reinforcement learning in the logistics industry: A case study at a large international retailer

November 2019

Author

Name	**M.W.T. Gemmink (Martijn)**
Programme	**MSc Business Information Technology**
Institute	**University of Twente** **PO Box 217** **7500 AE Enschede** **The Netherlands**
Email address	**M.W.T.GEMMINK@ALUMNUS.UTWENTE.NL**

Graduation committee

First supervisor	**Dr. Maria-Eugenia Iacob** **Department of Industrial Engineering and Business Information Systems** **University of Twente, Enschede, The Netherlands** **M.E.IACOB@UTWENTE.NL**
Second supervisor	**Dr. Marten van Sinderen** **Faculty of Electrical Engineering, Mathematics and Computer Science** **University of Twente, Enschede, The Netherlands** **M.J.VANSINDEREN@UTWENTE.NL**
Company supervisor	**Pieter Meints MSc.** **Logistics Support** **Albert Heijn, Zaandam, The Netherlands** **PIETER.MEINTS@AH.NL**
Daily supervisor	**Ing. Jean Paul Sebastian Piest MSCM** **Department of Industrial Engineering and Business Information Systems** **University of Twente, Enschede, The Netherlands** **J.P.S.PIEST@UTWENTE.NL**

Preface

Management summary

Whereas supervised and unsupervised learning has already reached widespread adoption within the logistics industry, reinforcement learning remains largely uncharted territory. Reinforcement learning is particularly interesting as agents can learn based on experience in a real-world or simulated environment. Current applications of the technique focuses primarily on games, however reinforcement learning could also be implemented within the business processes of logistic organizations. Because no clear and concise model for reinforcement learning adoption exists, this thesis is aimed at developing one. The main research question is therefore:

How can logistic organizations effectively assess and adopt reinforcement learning into their business processes?

Conducting exploratory research and a literature review formed the basis for a business process model aimed at logistic organizations in order to implement reinforcement learning. The exploratory research was an attempt to design and develop a reinforcement learning agent that could solve (a part of) the product allocation problem within the warehouses of Albert Heijn, also called slotting. The agent successfully learned how to allocate products according to the requirements as prioritized by the company. The insights of both the literature body and the creation of the agent were used to create the model to re-engineer business processes in the logistics industry using reinforcement learning.

The model was validated using expert opinions and the performance of the agent gives logistic organizations an idea about whether and how to use reinforcement learning in their business processes. The agent achieves high scores in the product allocation problem, but members of the Logistics Support department are still able to outperform the agent. Using intelligence amplification however, the cooper-

ation between the agent and the operational employees, the performance in terms of time and score of the slotting increased.

The contribution of this thesis to practice is that the model supports AI novice and AI ready departments within logistic organizations to re-engineer their business processes using reinforcement learning. Because these organizations have limited skills to implement a reinforced agent themselves, an example agent is provided that is ready to be used and experiment with. The scientific relevance is twofold. Current adoption models lack the unique determinants for artificial intelligence and reinforcement learning, the methodology of this research could alleviate this problem for future research. Secondly, this research also indicates that using intelligence amplification, agents using reinforcement learning also benefit from the cooperation between a human and the agent. The model can be considered a first step in taking reinforcement learning beyond simple games and towards actual business processes.

Acknowledgement

Utrecht, 15 November, 2019

Dear reader,

This thesis concludes my master Business Information Technology at the University of Twente. Little over 6 years ago I started my bachelor at this beautiful campus in Enschede and I've never regretted the decision to study at the UT ever since. Starting in 2013, the bachelor Business IT - featuring a valuable combination of computer science and management courses - brought me to where I am today. In those years I have been developing my personal, academic and professional skills. I have made a lot of friends during my time as a student and I cherish many unforgettable memories such as a study tour to South East Asia.

I would like to thank the people who were important during the writing of this thesis. First of all, I would like to thank my supervisors Maria Iacob, Marten van Sinderen and Sebastian Piest, for guiding me in writing this thesis and all the valuable feedback they have provided. As my daily supervisor, Sebastian always found the time to discuss my progress which really helped me in tackling issues and keep moving forward, thanks Sebastian! I would also like to thank Pieter Meints for his contribution and feedback during the project as my company supervisor at Albert Heijn. I always felt a member of the Logistics Support team and that really motivated me during the writing of the thesis. I really enjoyed having the opportunity to take a look at the logistic operations of Albert Heijn. Finally, I would like to thank my girlfriend, Niké, my family and friends for always supporting me throughout my studies. I could not have done it without them.

I wish you pleasant reading,

Martijn Gemmink

List of figures

List of tables

Abbreviations

A2C	Advantange Actor-Critic
AGV	Autonomous guided vehicle
AH	Albert Heijn
AI	Artificial intelligence
BPMN	Business process model and notation
CNN	Convolutional neural network
DC	Distribution center
DDQN	Double Deep Q-Network
DNN	Deep neural network
DOI	Diffusion of innovations
DP	Dynamic programming
DQN	Deep Q-Network
GUI	Graphical user interface
GPU	Graphics processing unit
IA	Intelligence amplification
IT	Information technology
LS	Logistics Support
LSP	Logistic service provider
MDP	Markov decision process
ML	Machine learning
NN	Neural network
RL	Reinforced learning
RNN	Recurrent neural network
SLR	Structured literature review
TAM	Technology acceptance model
TD	Temporal-difference
TOE	Technology-organization-environment
TPB	Theory of planned behavior
TRA	Theory of reasoned action
UTAUT	Unified theory of acceptance and use of technology
WMS	Warehouse Management System

Contents

I Initiation

1. Introduction

Modern artificial intelligence enables computers not only to solve problems based on human instructions but to solve them on their own [15]. Many believe that the future of AI is filled with potential and that it will become an important part of the logistics industry [6]. According to McKinsey the AI revolution is not in its infancy, but the majority of the economic impact is yet to come [9]. In recent years artificial intelligence has been studied intensively leading to a much better understanding of the technology. Artificial intelligence research has been around for 50 years and marketing has reached an all-time high [20, 26]. Because of modern computer power and large amounts of data, artificial intelligence is becoming increasingly interesting for logistic organizations that now can (partially) automate tasks that require a decent level of intelligence [6, 9].

"*Artificial intelligence (AI) is once again set to thrive; unlike past waves of hype and disillusionment, today's current technology, business, and societal conditions have never been more favorable to widespread use and adoption of AI.*" [6].

Almost everything we currently hear from in the field is thanks to deep learning. Deep learning works by using statistics to find patterns in data and it has proven to be successful in recent years. The sudden rise and fall of different techniques have characterized research for a long time and an analysis of more than sixteen thousand papers suggests the same could happen to deep learning in the near future. The research also identified upcoming trends in the field, one that keeps coming up is reinforced learning[1]. Reinforced learning gained momentum in October 2015, when DeepMind's AlphaGo defeated the world champion in a game of Go. With reinforced learning an agent is trained using punishments and rewards, much like how humans learn in the real world [19].

[1] Reinforced and reinforcement learning are used interchangeably throughout the thesis, but are the same.

AI has become more favorable than ever before because of Big Data, cloud computing and processing power. AI is becoming an integral part of the future of logistic organizations. AI has the potential to "fundamentally extend human efficiency in terms of reach, quality, and speed by eliminating mundane and routine work" [6]. Logistics is becoming an AI-driven industry and there are already many examples such as autonomous guided vehicles (AGVs), intelligent robot sorting, predictive demand, capacity planning and many more [6].

1.1 Background

This research has been conducted over an eight month period at the Logistics Support department of Albert Heijn. The department ensures that the processes in the distribution centers run smoothly.

1.1.1 Albert Heijn

The organization is named after its founder Albert Heijn (1865 – 1945). Albert Heijn took over the small grocery store of his father Jan Heijn in Oostzaan, a municipality and a town in the Zaanstreek, The Netherlands. A few years later Albert Heijn opened its second store in Purmerend and started with its own production companies which roasted coffee beans and baked cookies to be sold in the expanding number of stores. In 1927 the number of stores reached 107. Albert Heijn passed away in 1945 and three years later the company went public.

Albert Heijn wanted its stores to be accessible to both the wealthy and the poor, his motto was: "The everyday affordable, the special accessible." The mission of Albert Heijn is to offer all ingredients for a better life. Bringing good, safe, sustainable and healthy food to millions of customers. The stores have a wide range of high-quality items and friendly, helpful service, long opening hours and online ordering enable customers to shop for groceries around the clock. Albert Heijn (AH) is currently the largest and oldest food retailer in the Netherlands. Albert Heijn has more than a thousand shops across the Netherlands and another 40 in Belgium. The organization is owned by Ahold Delhaize.

1.1.2 Ahold Delhaize

Ahold Delhaize is the result of a merger in 2016 between the Dutch Ahold and the Belgian Delhaize. The headquarters of the organization is located in Zaandam, The Netherlands. The organization operates retail companies across 11 countries, employing over 372 thousand people in more than 6 thousand stores. Last year in 2018, the net sales were 62.8 billion euro. Every week 50 million customers are served at the supermarkets, drug stores, convenience stores and liquor stores in one of the 19 local brands of Ahold Delhaize, of which Albert Heijn is one.

1.2 Motivation

Whereas supervised and unsupervised learning have been studied extensively, reinforcement learning kept a low profile over the years. Recently reinforcement learning gained momentum due to breakthroughs such as defeating the world champion in a game of Go. There is not much literature connecting reinforcement learning to practice that goes beyond games and towards actual implementation in a large industry such as logistics.

1.3 Problem definition

Logistic organizations lack the tools to effectively identify whether (parts) of their business processes are suitable for reinforcement learning. But even when these processes are identified, the implementation is not as straightforward as supervised and unsupervised learning.

1.4 Research goal

This thesis aims at easing the adoption of reinforcement learning in the logistics industry with a clear and concise model that is on a business process level that helps these organizations to effectively implement reinforcement learning.

1.5 Research questions

Based on the problem statement the main research question that has been identified is:

RQ **How can logistic organizations effectively assess and adopt reinforcement learning into their business processes?**

To be able to answer the research question the following sub-questions have been formulated:

SQ1 **What is the current state of artificial intelligence and especially deep and reinforcement learning in the logistics industry?**

SQ2 **What are the most important artificial intelligence adoption models and frameworks in the logistics industry?**

SQ3 **Which types of business processes are suitable for reinforcement learning?**

SQ4 **Which steps help logistic organizations in successfully implementing reinforcement learning?**

SQ5 **To what extent can the developed model help logistic organizations in the adoption of reinforcement learning?**

1.6 Report contents

The structure of this thesis is build around the different phases of the Design Science Methodology of Wieringa [46]. First the background information on the two main topics, technology adoption and reinforcement learning is considered in the problem investigation. The exploratory research implementation of reinforcement learning at Albert Heijn is also considered in Part II. Part III and IV are part of the design cycle in designing and validation the treatment. Part V includes both the conclusion and the discussion. In Table 1.1 the part(s) and their relation to the research questions is depicted.

Question	Type	Methodology	Part(s)
SQ1	Knowledge	Problem investigation	Part II
SQ2	Knowledge	Problem investigation	Part II
SQ3	Design	Exploratory research / treatment design	Part II & III
SQ4	Design	Exploratory research / treatment design	Part III & IV
SQ5	Design	Treatment validation	Part IV

Table 1.1: The report contents

2. Methodology

The method of research will be based on the Design Science Methodology of Wieringa [46], which is about studying an artifact in context. The goal is to develop a model that helps logistic organizations to effectively adopt reinforcement learning. This design problem, according to Wieringa, can be formulated as follows:

Improve the adoption of reinforcement learning in logistic organizations by designing a model that is on a business process level in order to effectively utilize its potential [46].

The engineering cycle is a rational problem-solving process which contains the task to carry out design science research. The engineering cycle is depicted in Figure 2.1. The cycle provides a logical structure of tasks and tells us that in order to justify a treatment we must understand the problem [46]. In design science, only the first three tasks of the engineering cycle are performed, starting with the problem investigation.

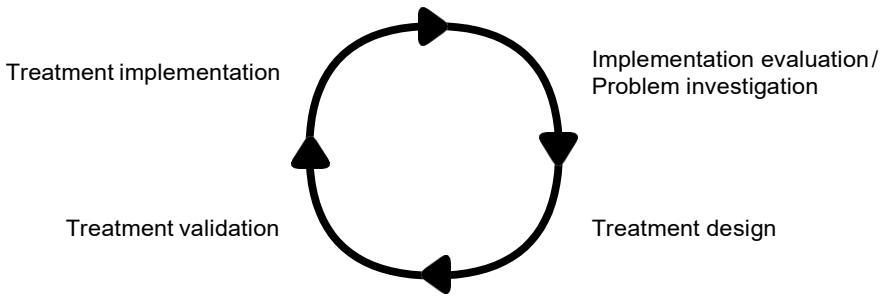


Figure 2.1: The engineering cycle [46]

For this thesis an approach will be taken that consists of the design cycle appended by exploratory research that has similarities to systems engineering, see Figure 2.2.

First a number of iterations are performed in an attempt to adopt reinforcement learning at the Logistics Support department of Albert Heijn, the largest food retailer in the Netherlands. The exploratory research together with a structured literature review will form a solid foundation for the problem investigation discussed in section 2.1. The next step is the treatment design, in which the requirements for the to be developed model are specified and the treatment(s) are discussed. The treatment design can be found in section 2.2. The final step of the design cycle is the treatment validation discussed in section 2.3.

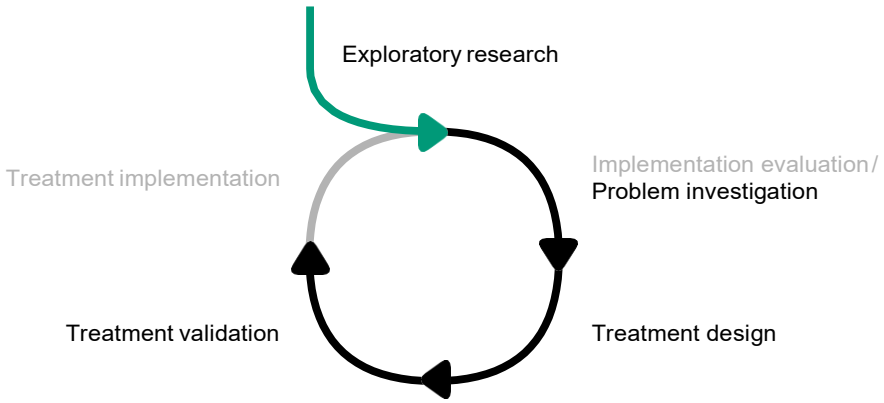


Figure 2.2: Research methodology

2.1 Problem investigation

The task is to investigate a problematic situation, starting with identifying, describing, explaining and evaluating the problem to be treated [46]. The problem investigation is twofold, both a structured literature review (SLR) found in section 2.1.1 and exploratory research in section 2.1.2 is considered. The goal of the exploratory research is to start the treatment design task with a strong literature foundation and the experience of actually carrying out a reinforcement learning adoption project at a large logistic organisation.

2.1.1 Structured literature review

This literature review aims to identify the problems, approaches, tools and applications of artificial intelligence and especially reinforcement learning as well as its adoption in logistic organizations in an attempt to identify what hinders progress in this regard. Both the scientific body as well as material from the logistics field will be considered.

An effective literature review creates a firm foundation for advancing knowledge [45]. First the literature search and selection will be discussed which also addresses the structured literature review and how the literature will be reviewed. For both main topics, a different search strategy was used.

Literature search and selection

Based on the research questions two main topics have been identified, reinforcement learning and technology adoption. Artificial intelligence is huge and during the last 50 years the field has become very disparate making it difficult to grasp [8]. The field of technology adoption and acceptance is on the other side of the spectrum being much more clear and concise. Because of the nature of the fields two separate methodologies were used. The specifics of each research method are discussed at the beginning of appendix A and B. The method of research for the structured literature review is based on the guidelines of Kitchenham et al. [22]. The two topics formed the basis for a systematic literature review (SLR). A SLR makes the review more valuable because it requires a legitimization for every choice made in the search process [47]. Before commencing with the review, first the sources have to be identified. The following sources will be used:

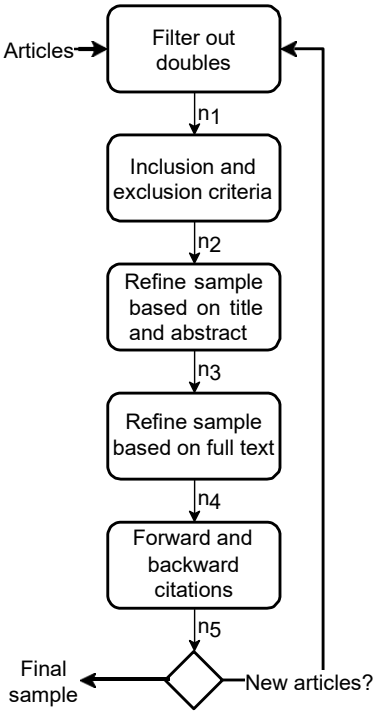


Figure 2.3: The literature selection process, based on Wolfswinkel et al. [47]

- Scopus WWW.SCOPUS.COM
- Web of Science WWW.WEBOFKNOWLEDGE.COM
- IEEE Explore WWW.IEEE.ORG/WEB/PUBLICATIONS/XPLORE
- Research Gate WWW.RESEARCHGATE.NET
- Springer Links WWW.SPRINGERLINK.COM
- Science Direct WWW.SCIENCEDIRECT.COM
- Google Scholar WWW.SCHOLAR.GOOGLE.COM
- University of Twente Library WWW.UTWENTE.NL/EN/LISA/LIBRARY

First Scopus and Web of Science were used for a preliminary search for the title, keywords and abstract. The selection of the final sample will be based on the selection process of Wolfswinkel et al. [47]. An iterative selection process that starts with filtering out the doubles. For every topic there will be inclusion and exclusion criteria that limits and improves the quality of articles found. From the remaining sample the title and abstract will be read and when relevant, the full text also. Forward and backward citations are used to evaluate the foundation on which the author(s) statements are based and to find more relevant articles. The literature selection process can be found in Figure 2.3.

Reviewing the literature

With the final selection of articles the next step is to review the literature and to identify the key concepts that arise. Webster & Watson recommend using a concept matrix when reviewing the articles, synthesizing the literature by discussing each identified concept. The concept matrix can be found in Table 2.1.

Articles	Concepts				
	A	B	C	D	...
1			**x**		**x**
2		**x**	**x**		
...				**x**	**x**

Table 2.1: The concept matrix by Webster & Watson [45]

Articles	Concepts				
	A	AB	B	C	...
1		**x**			**x**
2		**x**	**x**		
...				**x**	**x**

Table 2.2: The advanced concept matrix by Wolfswinkel et al. [47]

In order to expose potential relevant relations between concepts and their properties the concept matrix can be extended by merging concepts. Identifying what concepts to merge is a continuous process during the analysis. The advanced concept matrix proposed by Wolfswinkel et al. can be found in Table 2.2.

2.1.2 Exploratory research

The technology adoption models, combined with the specifications of reinforcement learning from literature will be the starting point of a small engineering cycle within the Logistics Support department of Albert Heijn. The goal of this exploratory research is to explore to what extent reinforcement learning can be adopted. The results of this exploratory research will be used as input for the model. The exploratory research consists of three phases.

Identifying suitable business processes

Based on the determinants of reinforcement learning and the puzzles it is able to solve one can identify which business processes are suitable for the technique. Three potential business processes will be identified based on unstructured interviews with employees of the LS department. One business process will be picked based on criteria defined before selecting the processes. The criteria are based on the literature body of RL.

Implementation of reinforcement learning

In this phase an attempt will be made to automate (a part of) the business process using reinforcement learning. Multiple experiments will be conducted to test different algorithms in order to get an understanding about what their advantages and drawbacks are in terms of performance and ease-of-use, starting with the most basic algorithm and scaling up from there. The implementation attempt will also give an idea about the performance of RL in a business process.

Adoption within the LS department

A single technical implementation is not sufficient for actual adoption, the organizational aspects of the adoption of reinforcement learning need to be considered. The aim is to determine what makes a logistic organization adopt a new technology such as artificial intelligence and in particular reinforcement learning. A logbook will be kept on all actions taken and whether or not they contributed to the adoption.

2.2 Treatment design

In this step of the design cycle the requirements are identified and how they contribute to the goals of the artifact [46]. The requirements are defined based on the experience gained by the exploratory implementation of RL in the LS department. The validity of the treatment design will also be assessed.

2.3 Treatment validation

The final step is the validation of the model. The aim of the validation is to "develop a design theory of an artifact in context that allows us to predict what would happen if the artifact were transferred to its intended problem context" [46]. The experimental research is also part of the validation. With the validation complete, an assessment can be made to what extent the model is able to help logistics organizations in adopting reinforcement learning into their business processes. And secondly to what extent RL is able to solve the problems it faces. Finally the limitations of the model and directions for future work are identified.

2.3.1 Single-case mechanism experiments

Single-case mechanism experiments are conducted for the exploratory implementation of a real-world business process at the LS department. These experiments will be carried out with multiple types of agents and environments to assess if the agents are able to perform in the business process identified in the exploratory research.

2.3.2 Expert opinions

Both the exploratory implementation of RL in a business process and the model itself will be validated by expert opinions. Employees of the LS department have the ability to imagine how the developed agent will interact inside the business process and what effects this would have. They will also validate whether the model could help the LS department to effectively utilize reinforcement learning into their business processes.

II Problem investigation

3. Reinforcement learning

Reinforcement learning is a field within artificial intelligence. Intelligence is our important ability to perceive, understand, predict and manipulate a world that is far more complicated than ourselves. AI is not only concerned with understanding but also with building intelligent entities. Definitions of AI can be categorized in four categories, see Figure 3.1. The top dimensions are about *reasoning* and the bottom ones address *behaviour*. The definitions on the left are concerned with human performance whereas the right ones address rationality. A system is considered rational when it does the "right thing", given what it knows. Russell and Norvig define AI as the study of intelligent agents that receive percepts from the environment and perform actions [35]. This chapter starts with the general concept of AI, the importance of deep learning and finally dives into reinforcement learning.

Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Figure 3.1: Definitions of AI in four dimensions [35]

3.1 Artificial intelligence

AI was first mentioned at a conference in July 1956, but research into the nature of intelligence goes back to the Greeks and other philosophers [8]. In the 1980s researchers were finding out that creating AI was more complicated than anticipated and many companies failed to deliver on their promises, leading to the so-called "AI Winter" [8, 35]. Recently due to the greater use of the scientific method in experimenting with and comparing approaches AI has advanced more rapidly. Sub-fields of AI are more integrated and AI has found common ground with other disciplines [35].

Deng et al. identified three main waves in the world of AI. The first wave in the 1960s was based on expert knowledge engineering - often symbolic logic rules - on very narrow application domains. The second wave which came around in the 1980s was based on machine learning or *shallow* learning due to the lack of abstractions [15]. AI has seen a large resurgence over the past ten years and deep learning - the current wave - is one of the most contributing factors [9]. This is visualized in Figure 3.2. Other important factors are big data and technological advances in creating *general AI* [48]. Currently we are able to create *narrow AI*, which is able to solve specific problems, *general AI* is able to solve multiple problems, like humans. The stage in which AI exceeds humans significantly *super AI* can be reached [6].

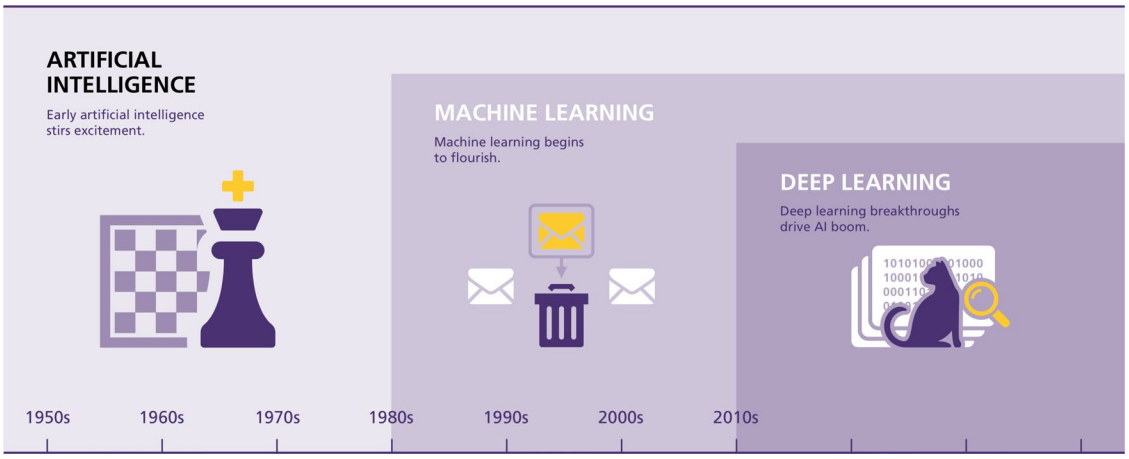


Figure 3.2: Artificial Intelligence overview [6]

3.1.1 Intelligent agents

Agents help in representing, analyzing, designing and implementing complex software systems [20]. According to Russel and Norvig: "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment trough actuators", this is visualized in Figure 3.3. The agent *percepts* inputs from its sensors and the history of what the agent has perceived is called the *percept sequence*. An agent's behavior is described by the *agent function* that maps any given percept sequence to an action. For complex problems this will be a very large - often infinite - table so often there is a bound to the length of sequences to consider. The *agent program* is the actual implementation of the agent function [35].

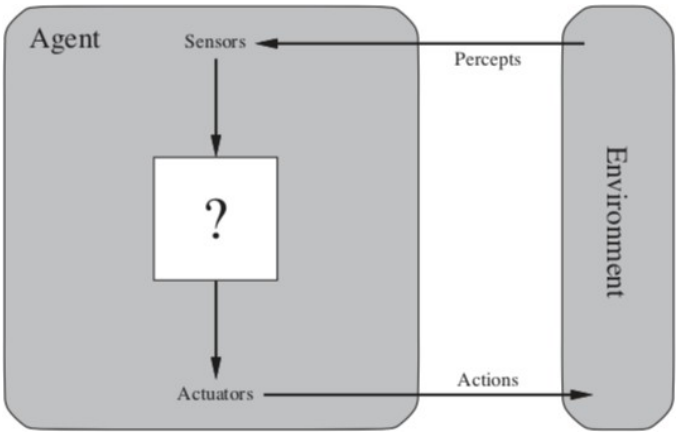


Figure 3.3: Agents interact with environments through sensors and actuators [35]

Rationality is an important concept in the book because it answers the question whether an agent is good or bad, intelligent or stupid. Whether an agent is rational is assessed by considering the consequences of the agent's behavior. The definition of a rational agent, according to Russel and Norvig:

"For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has" [35].

The environment states whether the agent's actions were rational. It is difficult to construct performance measures, both because "success" is often not clear. The authors state that "it is better to design performance measures according to what one actually wants in the environment, rather than according to how one thinks the agent should behave". Rationality is not perfect, because there is a level of uncertainty in the outcome. Omniscience is when the outcome is known beforehand, but this is impossible in reality. Agents sometimes have to perform certain actions to maximize the expected outcome, also called *information gathering*. In uncharted territory an agent might also perform some *exploration* in order to get familiar with the environment. The extent to which an agent is dependent on prior knowledge rather than its own percepts tells something about its level of autonomy [35].

3.1.2 Task environments

When designing an agent the environment needs to be specified as fully as possible. The authors define the *task environment* as the performance measure, environment, actuators and sensors. An example of a *performance measure* for a self-driving car is whether it is driving safe. The *environment* is the road, pedestrians and other traffic. The *actuators* can be the gas and brake pedal. Finally the sensors can be the cameras that register the road [35].

When describing a task environment the following dimensions need to be taken into account:

- Fully observable vs. partially observable: Whether the agent's sensors give the agent the complete state.
- Single agent vs. multiagent: If the performance of an agent is dependent on the behaviour of another, the task environment is multiagent. Agents can both cooperate and compete to a certain level.
- Deterministic vs. stochastic: When the next state of the environment is completely determined by the current state and the action by the agent, it is deterministic.
- Episodic vs. sequential: When an agent performs a single action and its actions are based on previous ones, it is episodic. An agent's short-term actions in a sequential environment can have long-term consequences.
- Static vs. dynamic: In static environments the environment does not change while an agent is considering an action. Dynamic environments continuously require the agent to take actions, even if it is still deciding.
- Discrete vs. continuous: When the environment has a finite number of states and potential actions, it is considered discrete. Continuous environments handle environments that have infinite distinct states.

3.1.3 Agent programs

Russel and Norvig identify four basic kinds of agent programs, each program is considered below.

Simple reflex agents

As the name implies this is the simplest type of agents. An agent selects actions based only on the current percept, ignoring the percept history. Based on sensor data and condition-action rules the agent takes actions. The schematic overview of a simple reflex agent is shown in Figure 3.4. Simple reflex agents work best when the task environment is fully observable [35].

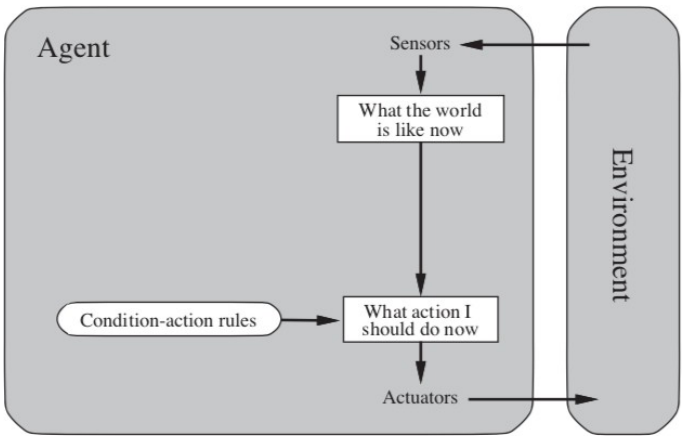


Figure 3.4: Schematic diagram of a simple reflex agent [35]

Model-based reflex agents

Model-based reflex agents can handle partial observability because they keep track of parts of the environment it cannot see. These agents maintain an internal state based on the percept history. The agent requires knowledge to be encoded into the agent program, how the environment evolves independently of the agent

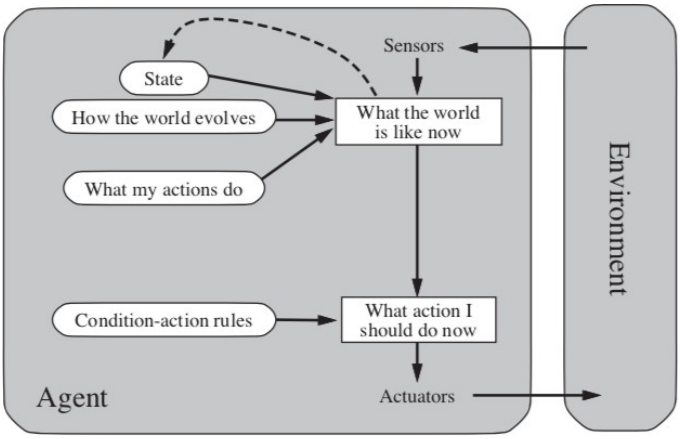


Figure 3.5: Schematic diagram of a model-based reflex agent [35]

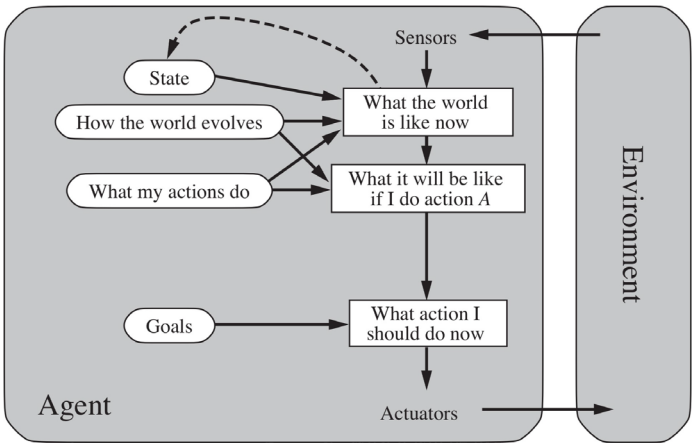


Figure 3.6: Schematic diagram of a goal-based agent [35]

and how the agent's own actions affect the world. A model is created that attempts to describe the environment on which the agent decides its actions [35]. The model-based reflex agent is shown in Figure 3.5.

Goal-based agents

Having knowledge about the environment is not always sufficient to know what to do. Here goal-based agents come into the equation. The agent has some sort of goal that help in deciding an action that is desirable. The goal can be straightforward when it is short term or immediately after an action but can be complex when it is achieved in the long run [35]. The schematic representation of a goal-based agent can be found in Figure 3.6.

Utility-based agents

In order to generate high-quality behaviour in most environments, goals are not sufficient. Considering rationality, the goal does not always justify the means. Utility-based agents therefore also take into account utility, which is essentially an internalization of the performance measure. When multiple actions result in the same goal or the goals are uncertain, a utility function can produce an appropriate trade-off [35]. The schematic overview of a utility-based agent is shown in Figure 3.7.

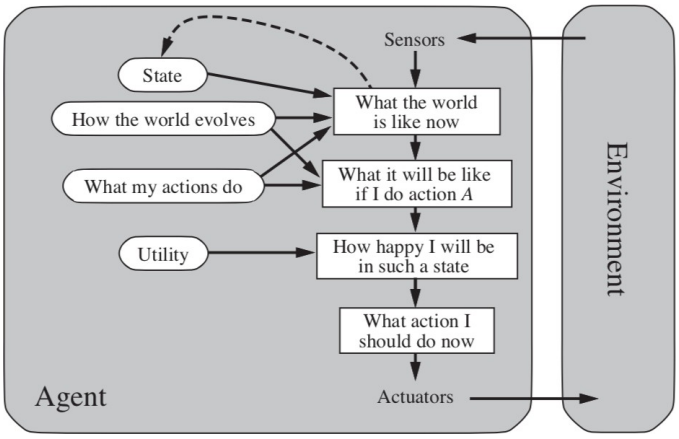


Figure 3.7: Schematic diagram of a utility-based agent [35]

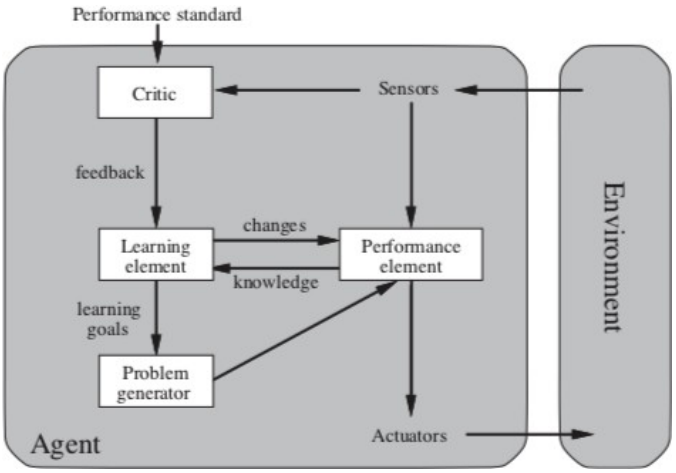


Figure 3.8: A general learning agent [35]

Learning

Agents can improve through learning. In creating state-of-the-art systems the preferred method is to build learning machines and then to teach them. Learning also has the advantage that it is allows agents to operate in unknown environments. The *learning element* is responsible for making improvements and the *performance element* is responsible for selecting actions, the previously considered agent. A fixed performance standard, called a critic, is used as an indication for the learning element for the agent's success. A learning agent could also have a *problem generator*, which suggests actions that lead to new and informative experiences. According to Russel and Norvig: "Learning in intelligent agents can be summarized as a process of modification of each component of the agent to bring the components into closer agreement with the available feedback information, thereby improving the overall performance of the agent" [35]. A general learning agent is visualized in Figure 3.8.

Representation of states and transitions

So far different agent programs have been discussed but not the representation of the state and its transitions. In an *atomic representation* each state of the world has no internal structure. A *factored representation* splits up each state

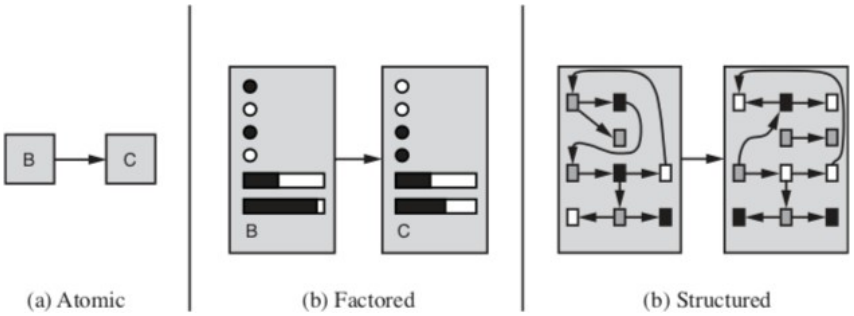


Figure 3.9: Representation of states and transitions [35]

into a fixed set of variables and attributes, each of which can have a value. In a factored representation states can share attributes. *Structured representation* is the most expressive of the three because it can explicitly describe various and varying relationships [35]. The representation of states and transitions in increasing expressiveness are shown in Figure 3.9.

Most of the time the more expressive language is much more concise, however learning and reasoning become more complex as the expressive power of the representation increases. "To gain the benefits of expressive representations while avoiding their drawbacks, intelligent systems for the real world may need to operate at all points along the axis simultaneously" [35].

3.1.4 Problem-solving

This section deals with the numerous ways in which agents can achieve its goals when no single action will do. Simple reflex agents cannot operate effectively in environments which are large and where it would take too long to learn. Goal-based agents consider actions and their outcomes however before searching for a solution, a goal as well as the problem must be identified. The decisions which the agent needs to make to reach the goal state is called the solution. An agent searches for the optimal (or most shallow) path towards the solution. There are numerous uniformed and informed search methods. Uninformed search is when only the problem definition is considered whereas informed search also considers the solution [35].

Searching for a solution works only for a single category of problems. When the problem is observable, deterministic in which the solution is a number of actions. When the problem is does not meet that requirements, different search techniques are needed. *Online search* is when an agent is faced with a state space that is unknown and must be explored [35].

In an environment in which an agent is trying to plan ahead and other agents are planning against us, for example in a game of chess, again other strategies are needed which work in competitive environments [35].

3.1.5 Learning techniques

There are multiple techniques to make an agent learn. Learning improves the agent performance on future tasks after making observations about the world.

Any component of an agent can be improved, the improvements depend on four major factors:

- Which component to improve.
- What prior knowledge the agent has.
- The representation of the component and its data.
- The feedback available to learn from.

There are three types of feedback that correspond to the three types of learning. Unsupervised learning means the agent is learning patterns even though no feedback is supplied, this often involves clustering. In supervised learning the agent observes inputs and the corresponding outputs and maps those in a function. The distinction is not always clear in real-world cases, e.g. semi-supervised learning in which is a combination of both supervised and unsupervised learning [35]. The final type of feedback is reinforcement learning, discussed in detail in section 3.3.

3.2 Deep learning

The "deep" in deep learning (DL) means that it uses one or multiple neural networks [35]. In this section neural networks are introduced as well as its importance in terms of recent developments.

3.2.1 Neural networks

As mentioned before a neural network consists of neurons (or nodes). A node takes inputs, performs some calculations and produces an output. An example of a node with two inputs x_1 and x_2 can be found in Figure 3.10 [35]. The calculation that happens in the example are:

1. Each input is multiplied by the weight. So $x_1 \rightarrow x_1 * w_1$ and $x_2 \rightarrow x_2 * w_2$.
2. The weighted inputs are added together with a bias b such that $(x_1 * w_1) + (x_2 * w_2) + b$.
3. Finally the sum is passed through an activation function in such a way that $y = f(x_1 * w_1 + x_2 * w_2 + b)$.

The activation function is used to turn an unbounded input into an output that has a predictable form. There are multiple activation functions but one of the most common is the Sigmoid function. The Sigmoid function only outputs numbers in the range (0, 1), it compresses values [35]. There are multiple activation functions, an overview of the most common can be found in Figure 3.11.

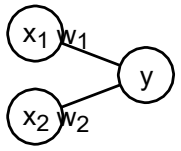


Figure 3.10: Representation of a node inside a neural network

A neural network (NN) consists of many connected neurons, each producing a sequence of real-valued activations. The first layer is called the input layer which is a number of neurons which get activated through sensors perceiving the environment. When the input neurons get activated the other layer(s) get activated using weighted connections from the previous layer. The *credit assignment* is the

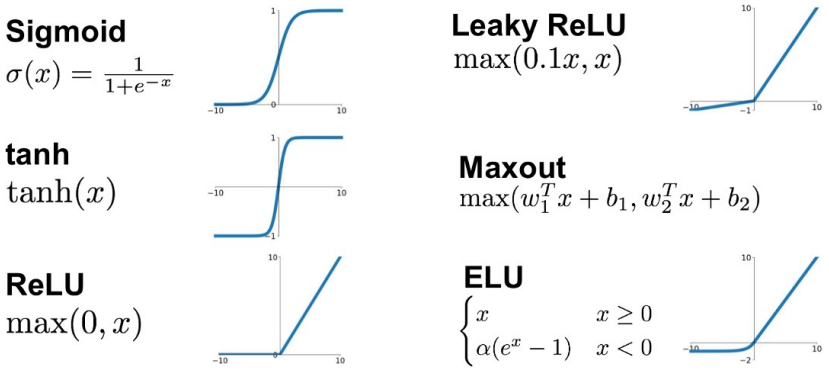


Figure 3.11: The most common activation functions

problem of finding the right weights that make the NN work properly. Deep learning (DL) is the process of creating NNs with many layers and accurately assigning credit to those layers [37]. An example of a neural network can be found in Figure 3.12. A hidden layer is any layer between the input and the output, the number of hidden layers can vary [35].

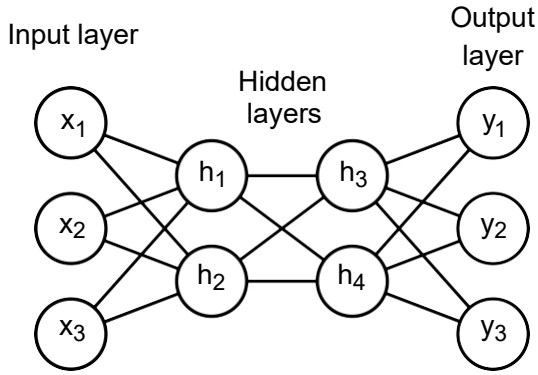


Figure 3.12: A neural network

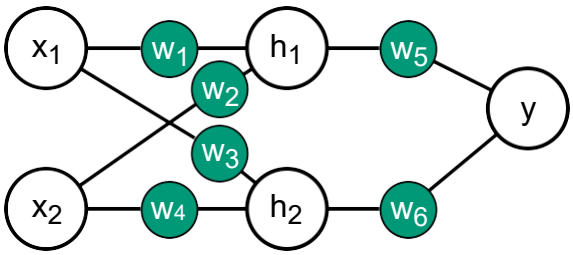


Figure 3.13: A small neural network including the weights

Imagine a small neural network with two nodes in the input layer, one node in the output layer and one hidden layer in between with two nodes. The resulting neural network is depicted in Figure 3.13. Consider $w = [0.5, 1, 0.8, 0.4, 0.3, 0.8]$, so $w_1 = 0.5$ and $w_6 = 0.8$. The processes of passing inputs forward in order to get an output is called feedforward. $b = [1, 0.1, 6]$ which are the respective biases in each node [35]. If we input $x = [4, 2]$ into the network with Sigmoid $s_a(x)$ with $a = 1$ as its activation

function the result can be calculated using the following equations:

$$\begin{aligned} h_1 &= f((w_1 * x_1) + (w_2 * x_2) + b_1) \\ &= f((0.5 * 4) + (1 * 2) + 1) \\ &= f(5) \\ &= 0.99 \end{aligned}$$

$$\begin{aligned} h_2 &= f((w_3 * x_1) + (w_4 * x_2) + b_2) \\ &= f((0.8 * 4) + (0.4 * 2) + 0.1) \\ &= f(4.1) \\ &= 0.98 \end{aligned}$$

$$\begin{aligned} y &= f((w_5 * h_1) + (w_6 * h_2) + b_3) \\ &= f((0.3 * 0.99) + (0.8 * 0.98) + 6) \\ &= f(7.08) \\ &= 0.99 \end{aligned}$$

When training a neural network, one attempts to minimize the loss. The lower the loss, the better are the predictions the network makes. A loss function often used is the mean squared error (MSE) [35]. The MSE can be denoted as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_{true} - y_{pred})^2$$

n = number of samples

y = the variable being predicted

y_{true} = the actual variable

y_{pred} = the predicted variable output from the network

With a clear goal of minimizing the loss we can write the loss as a multivariable function $L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$. When we want to tweak for example w_2 and want to know how the loss L would change we need the partial derivative $\frac{\partial L}{\partial w_1}$ [35]. Using the chain rule, this partial derivative can be written in the following formula:

$$\begin{aligned} \frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1} \\ &= \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1} \end{aligned}$$

The system of calculating partial derivatives starting from the back is called back-propagation. Using backpropagation one knows how to change the weights and biases in a network to make a better prediction. To train a network an optimization algorithm called stochastic gradient descent (SGD) is used that determines exactly how much the weights and biases need to change [35]. The update equation of

SGD looks like:

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

The learning rate is denoted as η which controls how fast we train. These steps are repeated for every sample we train on and slowly the network will improve [35].

3.2.2 The need for DL

DL already plays an important role in our lives and this is constantly increasing. Some of the application areas currently using DL are cancer diagnosis, custom made medicine, self-driving cars and forecasting. DL is about using optimizing techniques in order improve the accuracy and reduce the training time of neural networks. Shrestha et al. reviewed multiple optimization methods for different types of architectures. Their review includes convolutional neural networks (CNN), deep residual neural networks (DRN), recurrent neural networks (RNN) and reinforced learning (RL) [39]. The building blocks for simple NNs have been around for many decades but only recently they have attracted wide-spread attention by outperforming alternative methods. There are two types of NNs: *feed-forward* (FNN) and *recurrent* (RNN), both of which have been successful in the past. RNNs are the deepest of all NNs but also require much more powerful computers that FNNs because of their cyclic nature [37]. A feed-forward network has connections in one directions whereas a recurrent network feeds outputs back to its inputs. Neural networks are often used when more than one output needs to be considered [35].

When *shallow* neural networks were not capable in replicating human intelligence the machine learning community started focusing on DL [32]. It is not always clear when and if DL will outperform shallow NNs. Similarly there is no clear winner on which type of NN is best [39]. Poggio et al. reviewed and extended the theoretical literature about the conditions under which DL can be exponentially better than shallow learning [30].

An application in literature is enhancing transportation systems using DL. Wang et al. provide a comprehensive survey that focuses on the utilization of DL models to enhance the intelligence of the transportation systems. The authors identified which type of DL was best suited for the task at hand. Based on their results the authors identified a common pattern in applying DL models, starting with a simple DNN and slowly moving towards more sophisticated models. To reduce overfitting, a common problem on DL models, a useful strategy is to apply dropout which randomly ignores parameters during training [43]. Sze et al. wrote a review paper about the efficient processing of deep neural networks. DNNs deliver high accuracy on many AI tasks however the computational complexity and therefore its costs are high. The authors highlight important benchmarking metrics for practitioners to use [40].

DL has proven to be extremely successful however big challenges await. DL currently lacks interpretability and often require much more training than humans [15]. In the near future Deep Neural Networks will be able to - just like humans - actively perceive patterns by sequentially direction attention to relevant parts of

the data [37]. To tackle these problems both fundamental and applied research is needed, a new wave will not come without one or more breakthroughs in this regard. One of the potential breakthroughs Deng et al. mention is *deep reinforced learning* [15]. Garnelo et al. argue that a key objective for DL is to develop architectures capable of discovering objects and relations in raw data and to be able to represent them in ways that are useful for downstream processing [18]. During the next 5 to 10 years human level AI could be constructed, a thought based on thorough analysis of current rends in DL and brain reverse engineering [38].

3.3 Reinforcement learning

By using reinforced learning, an agent can learn what to do in the absence of feedback of a teacher. Without feedback, the agent does not know what actions are good and bad. Instead of learning an agent good and bad actions, one could also let it explore on its own and provide a *reward* when the agent attempts a good action. Rewards in environments can come at the end, immediately or anywhere in between depending on the problem. When playing a game of chess, it is difficult to reward individual actions but it is clear that checkmate is the goal [35]. Reinforced learning (RL) is a technique that can learn to predict consequences of behaviour in environments in order to optimize its actions [14].

"*The task of reinforced learning is to use observed rewards to learn an optimal (or nearly optimal) policy for the environment.*" [35]

Reinforced learning encompasses all of AI, an agent is placed in an environment and must learn to behave successfully. Reinforced learning can be passive, where the policy is fixed and the task is to learn the utilities of states. Another possibility is active learning, where the agent must also learn what to do. In order for an agent to gain a lot of valuable experience *exploration* is used. An example of exploring is when an agent takes an action it has not taken before to learn. An agent that explores more contributes to the learning and therefore increasing its rewards in the future. *Exploitation* is when an agent takes an action that - given its current knowledge - maximizes its utility [35]. RL is trained based on a simulation and therefore the underlying models used by most RL algorithms assume noise-free state information, whereas in practice the feedback is buried in noise and prone to delays [10]. Despite the difficulties deep reinforced learning enables scaling to problems that were previously unthinkable [3].

3.3.1 Core concepts of RL

In this section the core building blocks of reinforcement learning will be discussed, these concepts are used for RL algorithms and are not mutually exclusive.

Markov decision process

Reinforced learning is based on the Markov decision process (MDP) mathematical framework to tackle its problems. The MDP - introduced by Bellman in 1957 - produces an easy framework to model complex problems. The framework is used to model decision making in situations where outcomes are controlled partially random and partially by the decision maker [5].

A MDP is denoted as (S, A, P_a, R_a) [4], where:

- S is the set of states.
- A is the set of actions.
- $P_a(s, s^t) = Pr(s_{t+1} = s^t | s_t = s, a_t = a)$ is the probability that the action a in state s at time t will result in state s^t.
- $R_a(s, s^t)$ is the reward received by the transition from state s to s^t, by action a.

The problem of a MDP is to find an optimal policy. A function $\pi(s)$ that specifies which action to take in state s. The MDP in which an agent is interacting can be found in Figure 3.14. An action can be anything from a chess move or controlling a steering wheel. Rewards can be sparse, for example in a game of chess when they will come at the end or immediate in a game of pong. With sparse rewards it is often difficult to untangle what actions contributed to the final result.

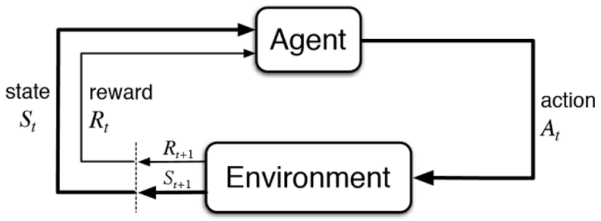


Figure 3.14: Reinforcement learning, derived from the MDP

In order to reward immediate rewards more than potential future rewards a discount factor γ can be used. The learning rate α is often used as a step size to determine to what extent newly acquired information overwrites old information. The horizon H tells us something about whether actions can take on forever or at a number of timesteps, it describes when the agent is finished [35].

Reinforced learning can solve MDPs without explicit specification of the transition probabilities. In reinforcement learning, instead of explicit specification of the transition probabilities, the transition probabilities are accessed through a simulator that is typically restarted many times from a uniformly random initial state [35].

Dynamic programming

Dynamic programming (DP) refers to a collection of algorithms that can be used to compute an optimal policy given a perfect model, such as a MDP [35]. DP uses a value function to structure and organize the search for good policies. Policy evaluation refers to the iterative computation of value functions for a given policy. And policy improvement is the computation of an improved policy given the value function for that policy. When combining these methods we obtain policy iteration and value iteration, the most popular DP methods. DP is not very practical for large problems, but are quite efficient for solving deterministic MDPs [4].

Monte Carlo methods

This learning method estimates value functions and discovers optimal policies without having complete knowledge about the environment. Mote Carlo methods only require experiences consisting of states, actions and rewards from interactions with the (simulated) environment. As Andrew et al. put it: "Learning from actual

experience is striking because it requires no prior knowledge of the environment's dynamics, yet is can still attain optimal behaviour". The methods are solving RL problems based on averaging sample returns. The underlying concept of Monte Carlo methods is to use randomness to solve problems that might be deterministic in principle. In off-policy methods, the agent also explores, but learns a deterministic policy that can be different from the policy followed. With on-policy methods the agent attempts to find the best policy that still explores [4].

Temporal-difference learning

Solving the underlying MDP is not the only way to tackle a learning problem. Another way is to use temporal-difference (TD) learning. TD is a model-free approach to learning how to predict a quantity that depends on future values. TD is a combination of Monte Carlo and dynamic programming (DP) ideas. TD methods can learn directly from raw experience without a model. TD methods learn based on estimates, "they learn a guess from a guess" also called boostrapping. Imagine updating Fridays weather forecast made on Monday when it is Wednesday and a much more accurate forecast can be made. With Monte Carlo methods one must wait until the end of an episode, because only then is the return known, whereas with TD methods one need wait only one timestep. In Monte Carlo methods in which experimental actions are taken, the learning is slowed down significantly. This is not such a big problem in TD methods because they learn after every action. Even though TD methods learn from immediate actions they still converge. It is still not clear whether Monte Carlo or TD methods converge faster but in practice TD methods usually converge faster [4].

3.3.2 RL approaches

There are numerous approaches used for RL, most can be categorized in the following approaches:

1. Model-based learning, use a model to find actions that have maximum rewards.
2. Value learning, estimating how good it is to take an action or reach a certain state.
3. Policy gradient, deriving a policy directly.

These approaches are not mutually exclusive but provide a way to classify the RL algorithms discussed in section 3.3.3.

Model-based RL

The idea of model-based RL is using a model and cost function to identify the optimal path of actions. A model predicts the next state after taking an action based on a model that is being optimized. Model-based RL agents are reflex-agents in which sensory input is processed and results in an action. Model-based RL has a strong competitive edge over other RL approaches because of its sample efficiency. The drawback however is that is is limited to the task it is designed for [35]. If physical simulation takes time, for example in robotics, model-based RL is a popular approach.

Value learning

This model-free method uses experience to learn directly based on state/action values or policies without the need of a world model [14]. Model-free methods are not as efficient as the model-based methods, because information from the environment is being combined with previous beliefs about state values, rather than being used directly. Sometimes it is difficult to determine which actions are responsible for the final result. For example when the rules of a game are clear, the optimal strategy cannot easily be determined. Using the Monte Carlo method one can calculate the value of a certain action or episode. Using DP the policy can be improved [35].

Policy gradient

As the name suggests, this approach focuses on the policy. Many human controls are very intuitive, without thorough planning finding the maximum return. Policy gradient works by making better rewards more likely to happen [4]. The reward function is defined as:

$$J(\vartheta) = \sum_{s \in S} d^{\pi}(s) V^{\pi}(s) = \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \pi_{\vartheta}(a|s) Q^{\pi}(s, a)$$

$d^{\pi}(s)$ is the stationary distribution of a MDP. Policy-based methods are very useful in continuous tasks. Using gradient ascent, one can move ϑ towards the direction suggested by the gradient $\nabla_{\vartheta} J(\vartheta)$ to find the best ϑ for π_{ϑ} that results in the highest return [4]. In order to compute $\nabla_{\vartheta} J(\vartheta)$ the policy gradient theorem is used, which simplifies the gradient computation:

$$\nabla_{\vartheta} J(\vartheta) = \nabla_{\vartheta} \sum_{a \in A} Q^{\pi}(s, a) \pi_{\vartheta}(a|s)$$

3.3.3 RL algorithms

Because of the huge array of RL algorithms in the literature only the ones used in this thesis are discussed.

Q-learning

In 1989 Watkins introduced Q-learning, which is a form of model-free reinforcement learning. Model-free reinforcement learning means that after learning, it can be viewed as a method of asynchronous DP. Q-learning enabled agents to learn how to act optimally in a fixed MDP by experiencing the consequences of actions without having to know the environment the agent is acting in. The learning process works by having an agent trying different actions in a particular state and evaluating the rewards and or penalties it receives, which is similar to the TD method. The rewards and penalties the agent receives can be infrequent and delayed. A long delayed reward can make it difficult to untangle the information and traceback what sequence of actions contributed to the reward [44].

The "Q" in Q-learning stands for the quality of an action taken in a given state. These states and their actions can be visualized in a so-called Q-table. An example of an initial Q-table with x states and 4 actions is depicted in Table 3.1 and after training in Table 3.2.

	a_1	a_2	a_3	a_4
s_0	0	0	0	0
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
s_x	0	0	0	0

Table 3.1: Initial Q-learning table

The algorithm calculates the quality of an action and updates the value inside the Q-table. At each timestep t the agents selects an action a_t and observes reward r_t and enters a new state s_{t+1}. The new Q-value is calculated using the following formula:

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) * Q(s_t, a_t) + \alpha * (r_t + \gamma * Q_{max}(s_{t+1}, a))$$

The formula uses TD learning to look one step ahead by taking into account the maximum Q-value in the next state s_{t+1} [44].

	a_1	a_2	a_3	a_4
s_0	0.1	0.3	6.4	-8.5
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
s_x	-2.3	0.4	5	-3.1

Table 3.2: Q-learning table after training

Deep Q-learning (DQN)

Q-learning is not very scalable and a different approach was needed to accommodate large and possibly even infinite state/action spaces. This is because the Q-table stores every state/action pair, imagine an environment with 10.000 states and 1.000 actions per state, the Q-table needs to hold 10 million cells. Here Deep Q-learning (DQN) comes in, because we can use a neural network to approximate the quality of an action [27]. To take advantage of the way neural nets work the Q-values are calculated for a specific state (called the Q-network), not a state-action pair. This is visualized in Figure 3.15.

DQN works by storing all past experiences of an agent in memory, determine the next action by the Q-network and by minimizing the loss function. The loss function in DQN is the mean squared error (MSE) of the TD part ($r_t + \gamma * Q_{max}(s_{t+1}, a)$) in the formula. Because R_{t+1} is the actual reward, the network - through backpropagation - is going to converge.

In order for agents to converge faster, a technique often used is experience replay. Which basically lets the agent reuse previous experiences in order to learn more from them. An important requirement for experience replay is that the laws of the

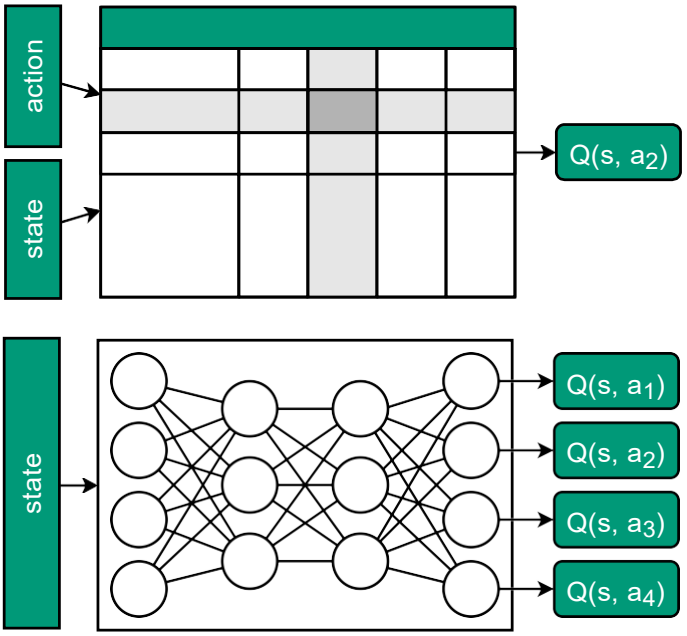


Figure 3.15: Differences between Q-table and the Q-network

environment do not change that result in past experiences becoming irrelevant [25]. Because some experiences are more valuable than others, a way to improve DQN agents is by using prioritized experience replay. Whereas in experience replay experiences are uniformly sampled from from the experience memory, prioritized experience replay attempts to replay important transitions more frequently. Schaul et al. extended DQN with prioritized experience replay and outperformed the standard DQN in 41 out of 49 games tested [36].

Although its great performance, DQN suffers from overestimating the Q-values. This is because the calculation of Q^{new} consists of its own prediction. It is therefore chasing a moving target which makes it slower to converge and because the prediction is based on the Q_{max} - the highest predicted next Q-value - it is overestimating. In an attempt to counter this problem, the Double Deep Q-learning algorithm (DDQN) was introduced by Van Hasselt et al. in 2016. The algorithm basically works with two neural networks, one for action selection and one for action evaluation. At an interval of n the evaluation NN is set equal to the action selection NN. Q_{max} is therefore not changing allowing for faster convergence. DDQN outperformed DQN on almost all 57 games tested by the authors [41].

Actor-Critic

Whereas policy gradient methods only update at the end of an episode, Actor-Critic methods uses TD learning to update at each timestep. This prevents that both good and bad actions are averaged as good when the final result is good. As a consequence policy gradients need a lot of samples. Actor-critic combines both policy gradients as well as the value function to increase efficiency [4].

The policy structure is known as the actor, because it is used to select actions, and the estimated value function is known as the critic, because it criticizes the actions made by the actor. Learning is always on-policy: the critic must learn

about and critique whatever policy is currently being followed by the actor [4]. This is visualized in Figure 3.16.

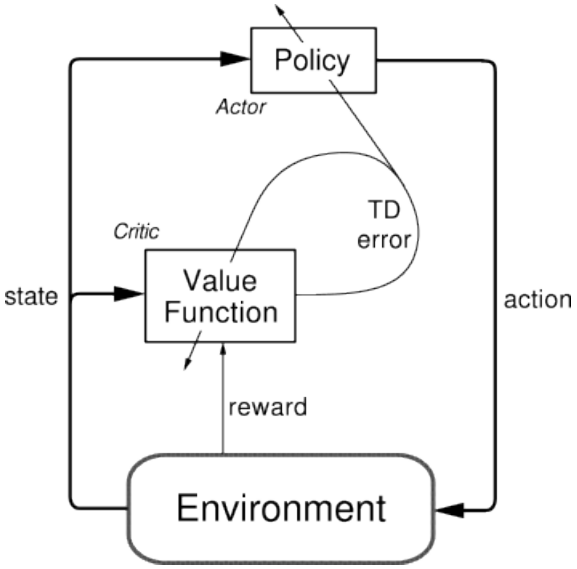


Figure 3.16: The Actor-Critic architecture [4]

Actor-critic methods consist of two models, which may optionally share parameters:

- Critic updates the value function parameters w and depending on the algorithm it could be action-value $Q_w(a|s)$ or state-value $V_w(s)$.
- Actor updates the policy parameters ϑ for $\pi_\vartheta(a|s)$, in the direction suggested by the critic.

Advantage Actor-Critic (A2C)

The advantage Actor-Critic (A2C) is a synchronous version of its asynchronous counterpart and is based on the Actor-Critic approach. A2C is an attempt to reduce the noisy gradients and the high variance of the basic actor-critic method. The actor-critic algorithm works with an advantage instead of the value function:

$$A(s, a) = Q(s, a) - V(s)$$

The advantage is the Q-value for a particular state minus the average value of that state. This function therefore tells us the improvement compared to the average action taken at that state. If $A(s, a) > 0$ the gradient is pushed in that direction, opposite when $A(s, a) < 0$. The advantage function can be estimated using the TD error, denoted as:

$$A(s, a) = r + \gamma V(s^t) - V(s)$$

A3C focuses on parallel training in which multiple actors are trained in parallel and get synced with global parameters after a number of timesteps [28]. Although being faster, because actors train independently on an "outdated" version of the global parameters the update is not as efficient compared to A2C. A2C also has

shown to utilize GPUs more efficiently and work better with small batch sizes while achieving similar or better performance than A3C [28].

3.3.4 Challenges of RL

Bus¸oniu et al. wrote a review that covers AI approaches to RL, from the viewpoint of the control engineer. The authors argue that stability is the main concern and that deep RL has been extremely promising in in recent years. This is because RL is particularly powerful in solving the function approximation and thus increasing accuracy [10].

There are numerous challenges in the field of RL from a neurobiology viewpoint according to Dayan et al. Model-free methods are less efficient than model-based methods but provide greater flexibility. RL agents focus on exploitation - using past experiences to optimize outcomes - however exploration could also be beneficial to find potential benefits that increase the long-term gain. This balancing act however is very difficult. The same holds for the degree in which new information should overwrite old information [14]. RL is not as stable as its supervised and unsupervised counterparts because it is presented with different experiences every time it is trained, its reproducibility therefore is low. Good practice therefore is to have fallbacks and check in place to verify the agent has learned the correct strategy [10].

4. Technology adoption

Information technology (IT) is considered to be an essential tool in improving the competitiveness of organizations. In order to utilize the potential benefits of IT it needs to be adopted [34]. The explosive rise in technologies revolutionized the way in which businesses operate and the pace at which these activities take place. These innovations continue to have a profound impact on business processes across the entire organization [26].

4.1 Adoption models

In this section numerous adoption models are discussed that can be considered the foundation of the technology adoption field.

4.1.1 Technology Acceptance Model (TAM)

In 1967 Ajzen and Fishbein introduced the Theory of Reasoned Action (TRA) based on the theory that a person's behaviour is a function of his behavioural intention [2]. TRA suggests that stronger intentions lead to increased effort to perform the behaviour, which increases the likelihood for the behaviour to be performed. Technology adoption is concerned with the action of using technology and which elements influence this usage. Information systems can only improve organizational performance if used, hence the importance of a technology acceptance model. TRA is general and not designed to apply only to computer usage behaviour, therefore Davis et al. proposed a model tailored to computer usage. found that resistance of these systems by managers and professionals is a big problem. The authors aimed at predicting people's acceptance of computers from a measure of intentions, therefore building on the theoretical basis of TRA. The intentions considered were attitudes, subjective norms, perceived usefulness and perceived ease of use. This resulted in the technology acceptance model (TAM) visualized in Figure 4.1 [12]. A comparison was made on the two theoretical models and

resulted in the identification of a more powerful causal structure in predicting and explaining user behaviour [13].

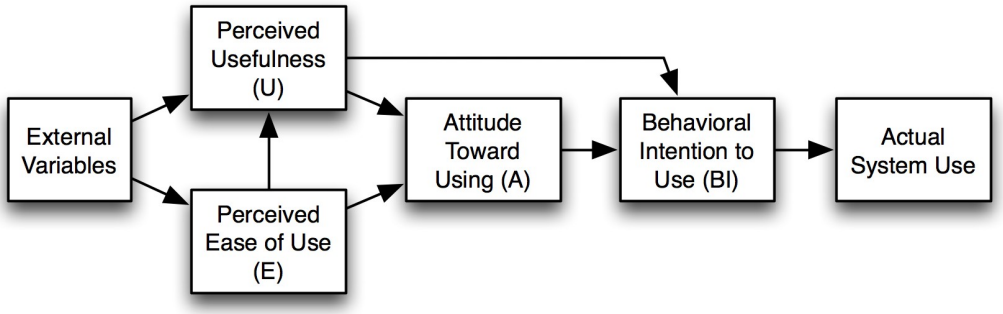


Figure 4.1: Technology Acceptance Model [12, 13]

In 1991 Moore et al. saw that in the technology adoption field there were a lot of mixed and inconclusive outcomes. The authors considered the various perceptions an individual may have towards an IT innovation [29]. Karahanna et al. conducted a study to find out whether or not pre-adoption and post-adoption beliefs are different. The authors also provide preliminary evidence suggesting that these are indeed not the same [21].

4.1.2 Diffusion of Innovations (DOI)

Another angle at explaining, how, why and at what rate new ideas and technology spread Rogers introduced the Diffusion of Innovations theory (DOI) [33]. Diffusion is the process by which an innovation is communicated through certain channels over time among the members of a social system. Diffusion is a kind of *social change*, a process in which the structure and function of a social system changes. When new ideas are invented, diffused, and are adopted or rejected, leading to certain consequences, social change occurs. Diffusion can be both planned and spontaneous but is often a combination of the two. Rogers distinguishes two types of diffusion systems, centralized and decentralized. When a small number of officials decide when and how to diffuse an innovation as well as to evaluate it, the diffusion system is considered to be centralized. When potential adopters are solely responsible for the diffusion of an innovation the system is considered to be decentralized [33].

There are four elements in the DOI, the first being the *innovation* itself which is "an idea, practice, or object that is perceived as new by an individual or other unit of adoption". New does not imply new knowledge, someone may have known about an innovation for some time but stills needs to develop an attitude towards it and potentially adopt it. The characteristics as perceived by the members of a social system are:

1. Relative advantage, the degree to which an innovation is perceived to be better.

2. Compatibility, the degree to which an innovation is consistent with existing values, the past and needs.
3. Complexity, the degree to which an innovation is perceived to be difficult.
4. Trailability, the degree to which an innovation may be experimented with on a limited basis.
5. Observability, the degree to which the result of the innovation is visible to others.

The second element are the *communication channels*, which are the means by which the messages get from one individual to another. The information-exchange relationship between the individuals determine whether and to what extent the source will transmit the innovation to the receiver. Examples of channels are Social Media for rapidly inform a big audience or a face-to-face exchange on an interpersonal level. Most individuals make innovation adoption decisions based mainly upon subjective evaluation, instead of a scientific evaluation. Primarily the very first adopters also use a scientific approach to evaluate an innovation. The transfer of an idea between individuals occur more frequently when they are alike, the degree of *homophily*. One of the most distinctive problems in the communication of innovations is that individuals are *heterophilous*. For example some people are more technically competent than others, often leading to ineffective communication. However, without some degree of heterophily no diffusion can occur, as there is no new information to exchange.

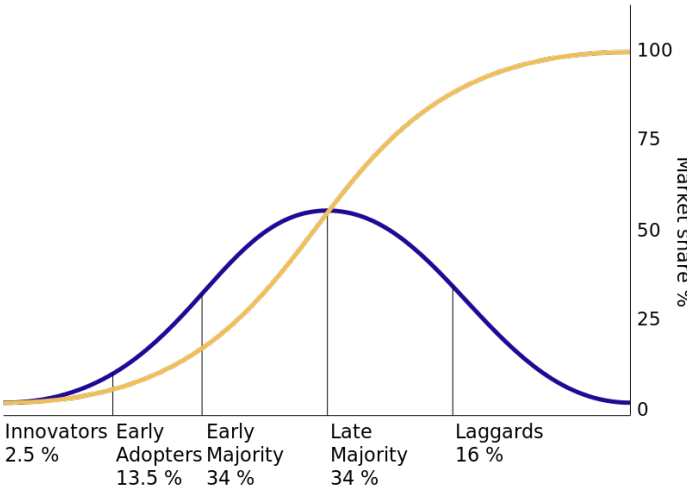


Figure 4.2: Diffusion of Innovations (DOI) [33]

Time is the third element of the diffusion process, often measured in how long it takes for a particular innovation to reach a certain amount of adopters. The DOI is visualized in Figure 4.2.

The final element of the DOI are the members of a *social system*, which is defined by Rogers et al. as "a set of interrelated units that are engaged in joint problem solving to accomplish a common goal". Diffusion occurs within a social system and the structure of the system constitutes a boundary in which an innovation diffuses. The structure allows one to predict behaviour to a certain degree and thus decreasing uncertainty [33]. Damanpour et al. used a sample of 1200 public organizations in the United States and found that organizational characteristics and top managers' attitudes toward an innovation have a strong influence [11].

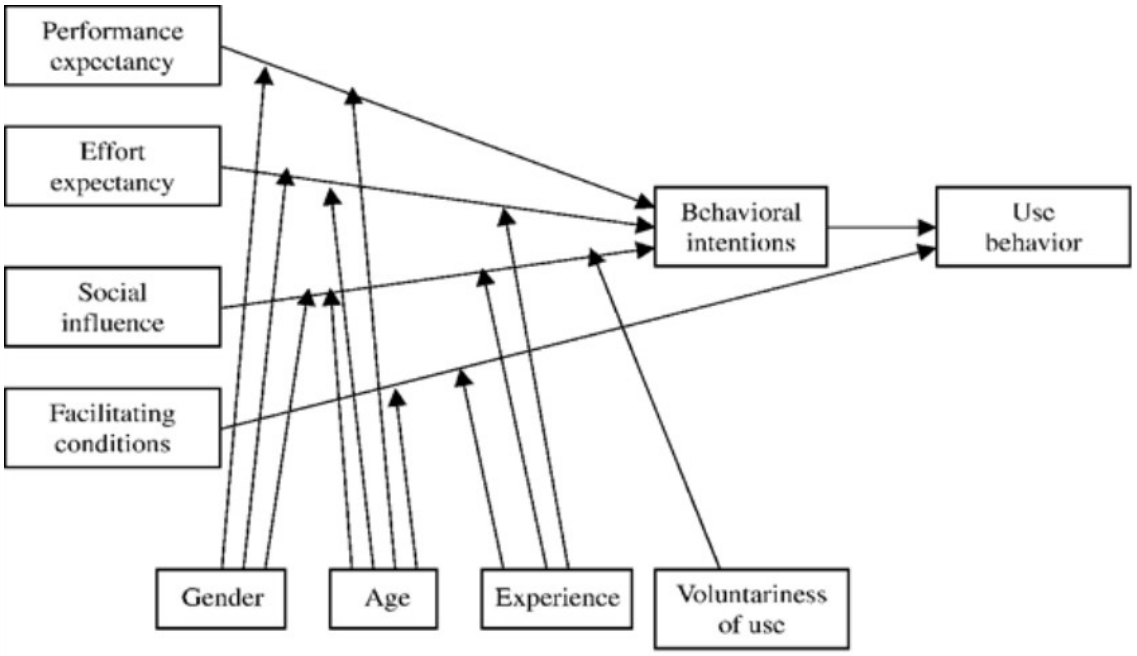


Figure 4.3: Unified Theory of Acceptance and Use of Technology (UTAUT) [42]

4.1.3 Unified Theory of Acceptance and Use of Technology (UTAUT)

TAM has been extensively studied and expanded but the most influential update was the Unified Theory of Acceptance and Use of Technology (UTAUT model considered next. In 2003, there was already a huge array of information technology acceptance models and Venkatesh et al. aimed at creating an unified view of those models [42].

The authors compared the following eight models; TRA, TAM, the motivational model, the theory of planned behaviour (TPB), a combination of TAM and TPB, the model of PC utilization, DOI and the social cognitive theory. The models were reviewed over a period of six months at four organizations in which their predictive power was assessed.

Venkatesh et al. formulated the Unified Theory of Acceptance and Use of Technology (UTAUT) which is visualized in Figure 4.3.

4.1.4 Technology-Organization-Environment (TOE)

The technology-organization-environment (TOE) framework was first mentioned in Torznatzky and Fleischer's *The Processes of Technological Innovation* in 1990. Whereas the book describes the entire process of innovation, the focus will be on the adoption chapter from DePietro et al. [16]. Since the book was published it remains among the most prominent and widely used theories of innovation adoption in organizations. The framework consists of three elements, the *technological context*, the *organizational context* and the *environmental context*. The contexts influence *technological innovation*, as shown in Figure 4.4.

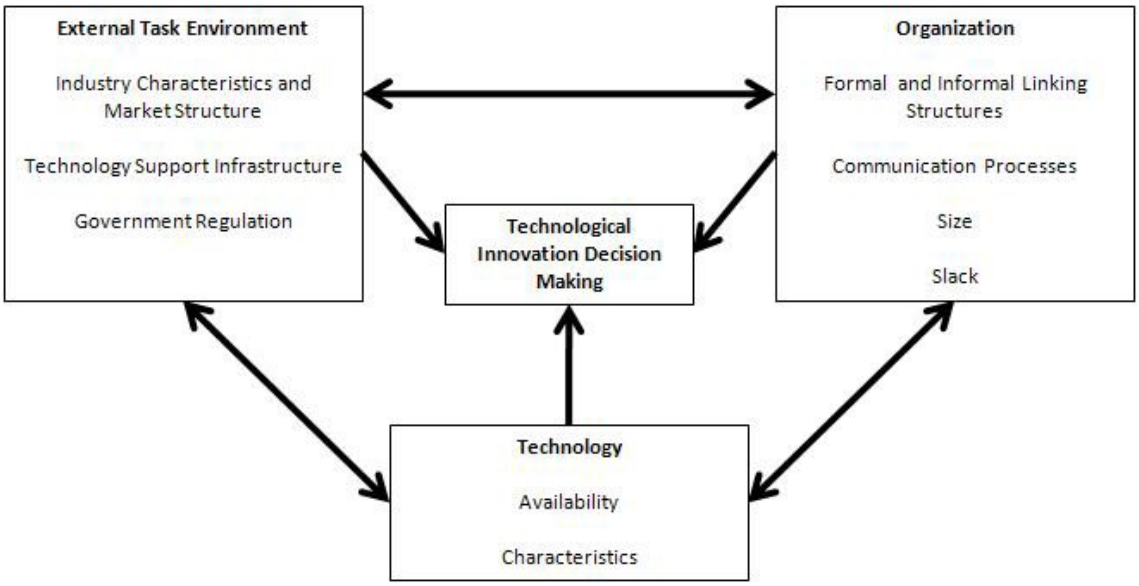


Figure 4.4: The technology-organization-environment (TOE) framework [16]

4.2 Intelligence amplification

In the core concept op AI, the machine mimics and replaces the cognitive abilities of the human brain. Dobrkovic et al. argue that there is a fundamental difference in the type of tasks intelligent agents excel at and the type of tasks humans do well. This difference is visualized in Figure 4.5. Dobrkovic et al. define intelligence amplification (IA) as "enhancing human decision making abilities through a symbiotic relationship between a human and an intelligent agent". When implementing AI in business processes it can be very helpful to identify the type of task and whether or not an AI agent is the way forward [17].

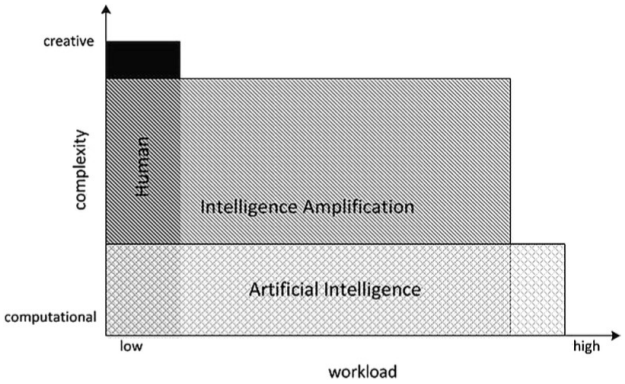


Figure 4.5: Decision making according to problem complexity and workload [17]

4.3 AI adoption in logistics

Not every innovation can be considered equivalent, an over-simplification sometimes made in the past [33]. This is especially true for AI because it has the potential to learn and develop intelligence that can imitate humans and solve complex problems. Whereas previous literature primarily focused on the adoption of technologies, the specific determinants of AI adoption are easily overlooked. Mahroof therefore aimed at exploring the barriers and opportunities of AI adoption conducting a case study at the warehouse of a major food retailer. The focus of the research was to explore the AI readiness of a large retailer from a human-centric

perspective. Using semi-structured interviews and TOE as the conceptual basis, the opportunities and barriers of AI adoption in the warehouse were identified. TOE is often extended with 'perceived benefits' which refers to the relative advantage that AI technology can provide to the organization. According to Mahroof, the TOE framework with the extension provides the ideal lenses for assessing AI readiness [26].

A part of the adoption is the potential to deliver the perceived benefits. Leung et al. used AI to support decision makers in generating wave picking sequences to handle e-commerce shipments. Considering that order picking is one of the most costly activities in a warehouse, efficient wave picking was crucial for reducing the costs. A pilot study at a logistic service provider (LSP) showed that the order processing efficiency was greatly enhanced [24].

Klumpp et al. presented a literature review on the development of AI applications and outlined potential risks to social sustainability of an artificial divide between employees and the organization. The conclusion is that in order to get a fully automated supply chain there needs to be sufficient attention for the human interaction factor [23].

"*In summary, the future competitiveness and logistics performance will significantly depend on the described factors regarding human work motivation as well as human–machine cooperation and acceptance.*" [23]

4.4 AI in practice

DHL and IBM have jointly written a report on AI in logistics in which the impact on logistical organizations is assessed, as well as best practices from other industries that can be applied to logistics [6]. Accenture wrote a report on AI maturity and models for success [1].

It is not a simple task to shift current logistics operating models to models which incorporate AI. Because of the abundance of machine learning methodologies it is difficult to find the right one for the problem at hand. DHL and IBM created an overview to aid the search which is visualized in Figure 4.6.

It is useful to identify what business problems could be solved by, and demands AI. When a business problem that can be improved by AI is identified, the next step is to cautiously assess the potential value drivers such as cost reduction and an improved customer experience. Before commencing with the implementation the available and required data has to be considered as well as how clean it is, how often it is collected and how it can be accessed. Depending on the project time span and the AI skills available in the organization it may be better to start a one-off project instead of a long-term initiative that requires changes in the core of the organization. When AI skills are lacking within an organization, it could be necessary to outsource projects [6].

In their report DHL and IBM also identify two types of AI projects, cost reduction and

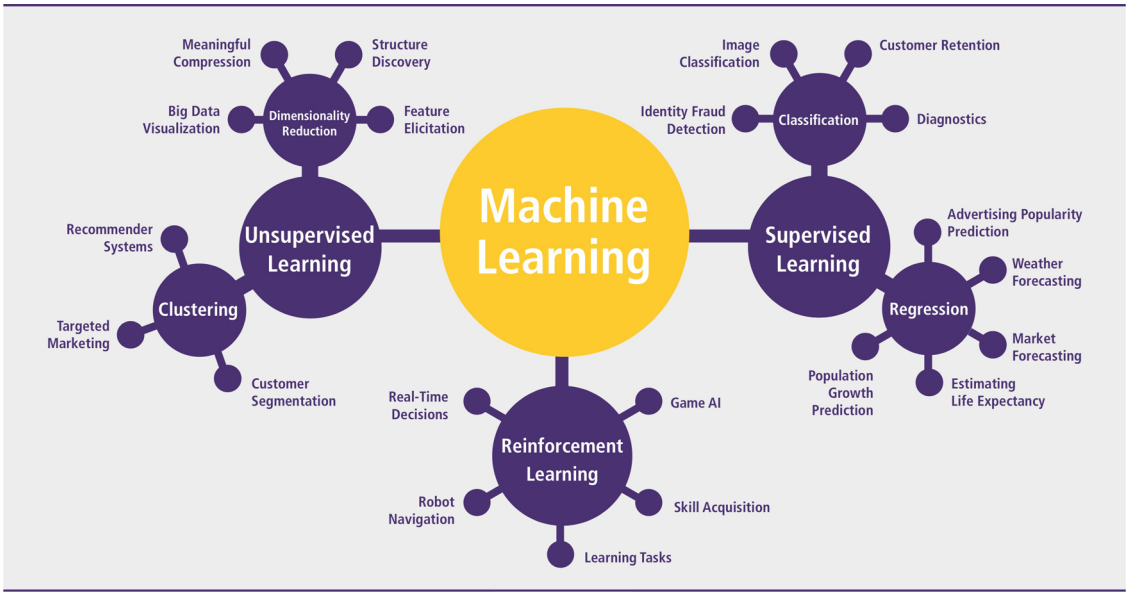


Figure 4.6: Machine learning taxonomies [6]

new value creation. For cost reduction projects they provide a framework which can help in deciding whether a project is a classical analytics project, machine learning project or an AI + human project. The cost reduction decision tree for AI projects can be found in Figure 4.7 [6].

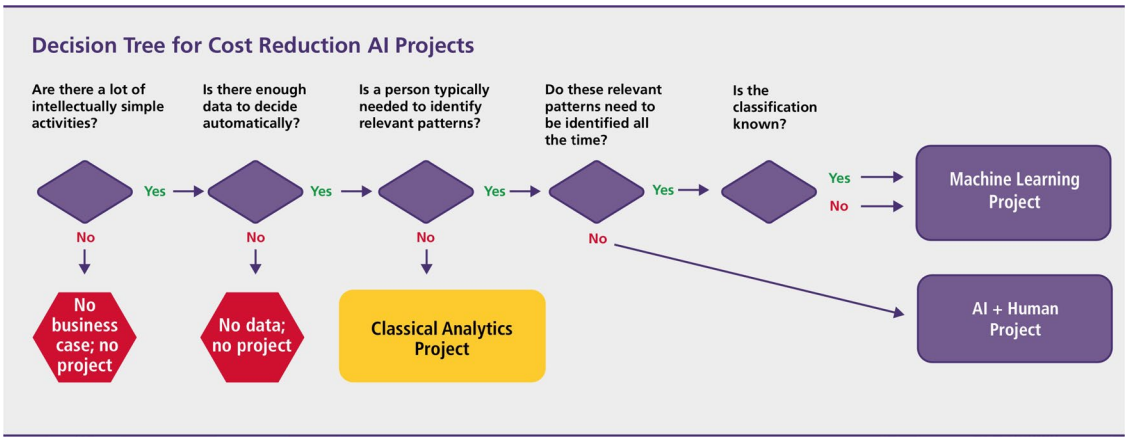


Figure 4.7: Decision tree for cost reduction [6]

The culture of an organization is also a big factor in the adoption of AI. The adoption and its required cultural shift could be more difficult than the technical implementation challenges. Job loss is often the biggest fear and having leaders in an organization that are supportive of new technologies and who are able to bring change is crucial for success [1, 6].

An agile approach in an organization enables employees to learn and evolve with new AI systems. Data is important and the initial quality of the AI agents can be lower than currently however over time the agent has the potential to surpass it. The question that arises is what quality standard do we accept for it be

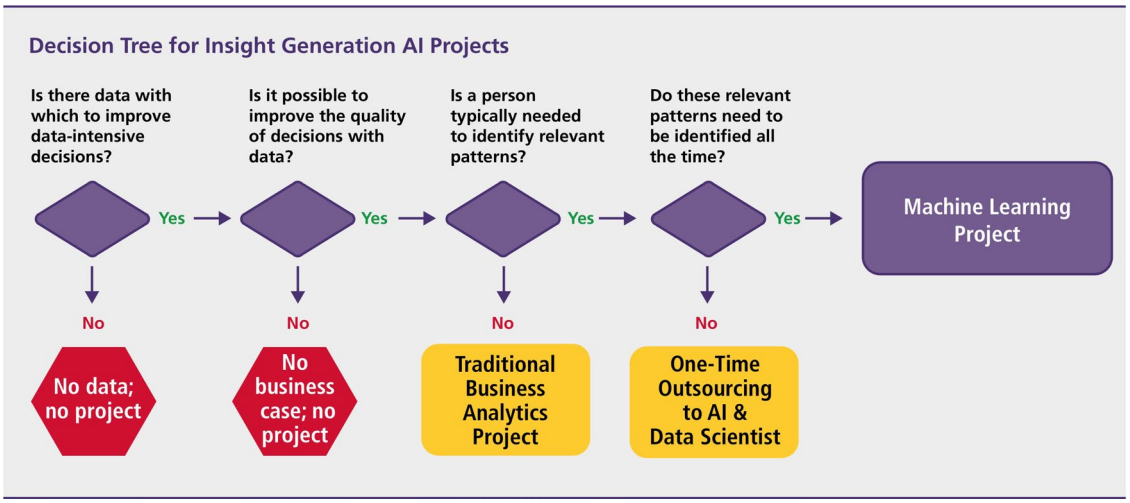


Figure 4.8: Decision tree for insight generation [6]

economically viable [6].

When an approach is identified the next step is the actual implementation. There are numerous ways to carry out the implementation, some recommendations from DHL and IBM are:

- Design thinking to reveal unmet needs of users.
- Traditional IT project management to scope the system(s).
- AI-specific methodologies for creating and training the models.
- Agile methodologies for continuous development and improvement after deployment.

According to Accenture nearly 20% of leaders identify resistance from employees due to concerns about job security [1].

4.5 Maturity models

When organizations want to know how far they are in terms of becoming an "AI-first" organization, an assessment is often made using a maturity model. Numerous models and assessment techniques exist online, some of them are discussed in more detail here.

In their report Accenture describes emerging best practices regarding the implementation of AI. When considering the more successful AI deployments the most contributing factors are:

- Reviewing AI output on a weekly basis.
- Making sure there are processes in place to override questionable results from AI agents.
- Anticipation that more than 25% of processes being improved by AI in the next three years.
- Conducting ethics training.

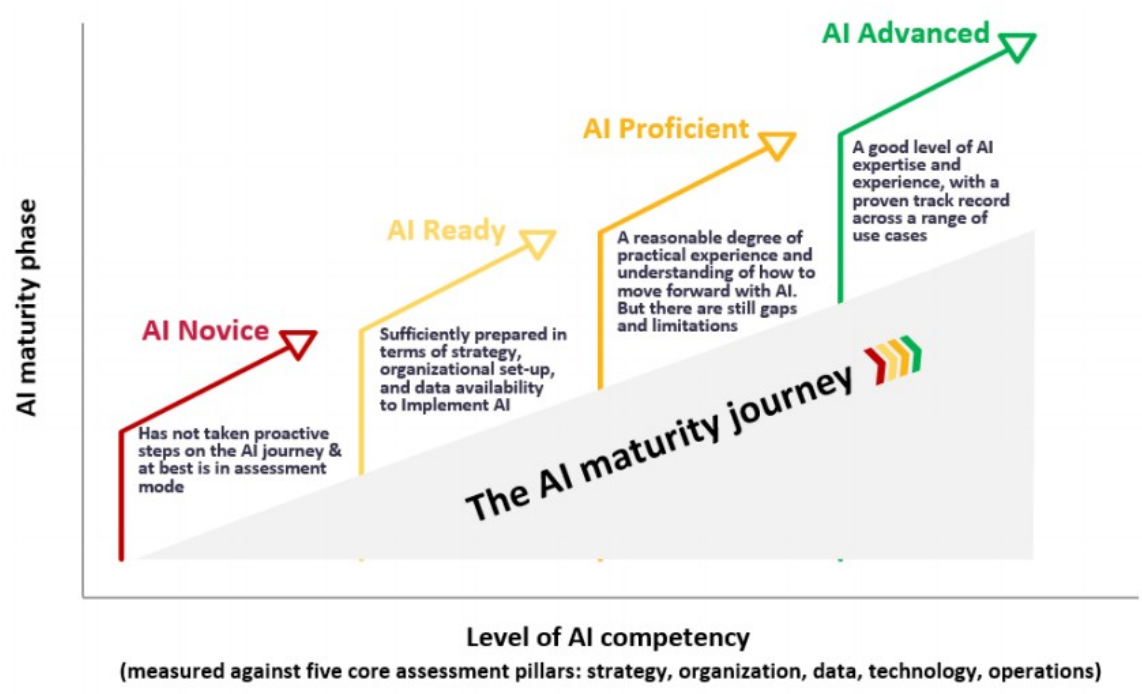


Figure 4.9: Level of AI competency [31]

- Connecting analytics to AI.
- Having faith in AI agents.

Based on the survey of Accenture there are no clear guidelines for success in AI, but that it is crucial for successfully managing the powerful potential of AI [1].

Other maturity models can be found online such as the model of Pringle et al. as shown in Figure 4.9 [31]. The model was written for communication and media service providers but the idea could be applied to the logistics industry. The authors identified the core pillars and assessment criteria needed for an AI maturity model, including strategy, organization, data, technology and operations. These core pillars contain a detailed set of criteria and associated questions designed to assess the AI maturity [31].

Whereas there are numerous maturity models and AI readiness scans to be found on the world wide web, the number of models tailored towards the logistic industry remains limited. Organizations have no clue whether or not a model is suited for their industry. The models are often very high level, useful for setting and managing goals, however they lack a clear method when starting to implement AI and identify potential low-hanging fruit.

5. Exploratory research

The literature review on reinforcement learning and technology adoption is important when creating a model, however having actual implementation experience is even more valuable. As Benjamin Franklin once said: "Well done, is better than well said". In this research an attempt will be made to implement reinforcement learning at the LS department. First the background is discussed in section 5.1 to get an idea about the supply chain and the department, as well as its readiness towards AI. Multiple business processes are identified in section 5.3 and for each the applicability of RL is discussed. Finally the actual implementation is discussed in terms of the organizational and technological aspects. This exploratory work helps in establishing priorities, establish definitions and together with the literature review forms the starting point for the treatment design.

5.1 Background

In this section the most important stakeholders within the supply chain are discussed.

5.1.1 Replenishment

This department is responsible for ensuring product availability in the stores by planning and controlling the flow of goods. Flow managers make sure that the supply chain operates smoothly. They make agreements with manufacturers, suppliers, logistic organizations and internal departments to reach this goal. The planners within the replenishment department constantly monitor the automatic orders and act when needed.

5.1.2 Distribution centers

Albert Heijn has several distribution centers (DCs) across the Netherlands. The national DC (LDC) is located in Geldermalsen and contains around 12 thou-

sand products such as slow-moving products, tobacco, dangerous goods and medicines. There are also four regional DCs (RDCs) which contain close to 4 thousand products which include fast-moving products including cooled products. There are also six DCs outsourced to logistic service providers (LSP). The remainder of product types such as (slow-moving) cooled products, flowers, non-food and frozen products are managed by those LSPs. The overview of DCs in the supply chain of Albert Heijn can be found in Table 5.1.

Type	Abbreviation	Operator	Location
National DC Regional DC	LDC DCP DCT DCZ DCO	Albert Heijn	Geldermalsen Pijnacker Tilburg Zaandam Zwolle
Shared Fresh Center XPO Oss (non-food) Frozen products	SFC	XPO Logistics	Nieuwegein Oss Hoogeveen
Shared warehouse cheese	SWK	Bakker Logistiek	Zeewolde
Returns		Kuenhe + Nagel	Pijnacker Tilburg Zaandam Zwolle
Flowers		GIST	Bleiswijk

Table 5.1: Locations and types of DCs of Albert Heijn

5.1.3 Logistics Support

The logistics support (LS) department of Albert Heijn is responsible for ensuring that the processes in the distribution centers run smoothly. The main responsibilities of the team, among other things, are:

1. Making substantiated decisions about where to place certain types of racks within a warehouse.
2. Slotting products within the warehouse for a smooth picking process.
3. The distribution of articles to stores in a way that is beneficial for both stores and DCs.
4. Ordering different types of load carriers to make sure there are no bottle-necks.
5. Ensure that pick orders are in the WMS in time.
6. Monitoring forecasts for the coming days/weeks or even months to make sure the DCs are prepared.

5.1.4 Transport

The department of transport is responsible for facilitating the transportation of products to distribution centers and shops.

5.1.5 Stores

The stores are responsible for selling products to the consumers of Albert Heijn. The shops make sure that products are in the shelves in time.

5.2 AI maturity at the department

The LS department has yet to start with artificial intelligence and therefore could be ranked "AI novice" in the maturity model of Pringle et al. [31]. Within other departments of Albert Heijn AI has already been successfully implemented such as personal discounts based on your personal buying behaviour and dynamic pricing to reduce wasting products that almost reached their best before date. In order to increase the awareness within the department and to be able to generate ideas suitable for AI and RL a team day was organized around this theme. To build upon that day a series of presentations was held to show the potential of RL and how the department could adopt this technology.

5.2.1 Team day at the university

On Friday the 24th of May the entire logistics support department of Albert Heijn went to the University of Twente for its annual team day. The team day is about learning something having some fun along the way. This time it was about inspiring colleagues into the world of artificial intelligence and other innovations such as drones. It is a great way to reach the entire audience including slotters, planners, managers and IT experts to get the dialog about AI started.

Sebastian Piest gave a presentation about what artificial intelligence is and its possibilities. This gave the team members a first impression of the technology and its capabilities. After the presentation the team members were first asked about potential uses of AI in the warehouses and logistics of Albert Heijn, this was done individually. After that the employees split into different groups to discuss their ideas and to also think about the consequences that AI could have on the department and the employees in particular.

Interesting questions that arose were:

- Where is the line between AI and an advanced algorithm?
- As the middle-man between transport and replenishment, with an AI agent making the decisions, are we still able to substantiate its decisions?
- If we rely on an AI agent to make important decisions, how can we make sure we are able to intervene when needed?
- Will AI replace or change my job?
- How do we get the knowledge to an adequate level to be able to maintain and improve the agents?

This event was an opportunity to identify opportunities and concerns of the different stakeholders at the logistics support department. The questions formulated above could hinder the adoption and tackling those is extremely important for the adoption of AI. The ideas that the members came up with for the logistics support department were:

- Automated slotting

- Indoor real-time positioning with dynamic orders
- Smart voice-picking using conversations
- Automatic stacking quality assessment
- AGV transito
- Automatic inbound classification
- Using drones for safety and automatically checking the inventory levels
- Store demand forecast automatic outlier detection
- Automatic dock assignment, a smart control tower
- Creating a virtual distribution center, where realistic simulations can be run

The ideas were discussed in the groups and this automatically led to some additional concerns:

- Do we want to be an 'owner' of an intelligent agent?
- Do we still go to the sites if decisions are made by an agent?
- What is the right moment to get into AI? Developments are going fast.

5.2.2 Demonstration agent

During a weekly meeting on the Friday afternoon the entire department is brought up-to-speed with the newest developments within the supply chain of Albert Heijn. In order to answer some of the concerns of team members and bring the AI discussion closer to the department I decided to give a demonstration of a simple reinforcement learning agent that was able to solve a small slotting puzzle. The demonstration was based on a Q-learning algorithm which is not scalable but it helped the team members to look beyond the state-of-the-art examples of AI and closer towards their own work. Because the department knew the potential applications of RL a suitable business process could be identified.

5.3 Identifying a suitable business process

The operational staff at the Logistics Department is most likely to perform repetitive and labour-intensive activities. Therefore a selection of those employees were asked about activities they perform often that could be suitable for reinforcement learning. Based on those talks three important business processes that could be suitable for RL in the Logistics Support department of Albert Heijn were identified:

1. The estimation of order pickers needed for production per shift.
2. The slotting process in the warehouses.
3. The identification of optimal locations for racks in a warehouse.

Each process was then ranked based on the following criteria:

- How crucial is this process for the operation?
  - Is there a fallback in case the agent does not perform?
- How labour-intensive is the task in daily operations?
- To what extent does the task environment fit a RL approach?
  - Can the task environment be simulated?
  - How big is the state-action space?
  - Are the rewards sparse, immediate or somewhere in between?

  - How effective should the agent be?
- What are the potential gains in terms of effectiveness given the current situation?

The criteria are based on the strengths and weaknesses of reinforcement learning. A reinforced learning agent is not as stable as its super- and unsupervised counterparts and therefore it is valuable to assess the importance of the process. If there is a fallback in place this could alleviate this problem to some extent. If a task is very labour-intensive an agent could save the employees a lot of time and thus costs making the potential rewards worth the risk. The task environment says something about the fit with reinforcement learning as well as the complexity of the problem the agent attempts to solve. In the next sections each case will be evaluated based on these criteria.

5.3.1 Estimating the number of order pickers per shift

Order picking at the DCs of Albert Heijn requires proper planning to make sure the orders are ready for transport in time to (a regional DC or) the supermarkets across the country. Because this operation runs 24 hours a day, order pickers can also work during the night. A RL agent could decide the most optimal ratio of day night workers. This is important because having to many order pickers at any time leads to congestion in the warehouse and thus to lower productivity, working during the night however is more expensive. This business process however is difficult to simulate as the penalties (such as congestion) can be the result of various external factors. The state-actions space depends hugely on what variables one takes into account, but selecting those variables is not straightforward. Making this planning is not very time-consuming as the operational staff already has tools that ease the decision-making process. This problem is not particularly suited for RL, a supervised learning agent could also be used to predict the productivity based on historical data.

5.3.2 Optimal rack locations

There are numerous location types available in the warehouses of Albert Heijn, such as regular pallet locations and flow racks. Flow racks consists of multiple smaller locations that are great for products that are not sold very often, this reduces the required area that the product occupies and therefore the distance the order pickers have to travel to complete their order. Deciding where which type of location goes is a complex task, having to many flow racks at the beginning of the circuit limits the number of products allowed in those locations. The major downside of this business process is its occurrence frequency. Only when a (part of a) DC is remodelled, the rack locations are identified and therefore the agent is needed. Similar to estimating the order pickers per shift, this problem is difficult to simulate. The layout of the DC needs to be combined with the products that have to be allocated and this is based an a lot of assumptions that could alter the results. The actual results of the agent are also difficult to measure, as the performance of order pickers after a remodeling does take some time. Order pickers need to get accustomed to the new layout and tuning and improving the agent would take a lot of time. Although an RL agent could be used for this problem, it is unsuitable for an initial implementation that requires a lot of iterations.

5.3.3 Slotting

Slotting is the process of organizing inventory in a warehouse or DC. It is an important tool to create an effective warehouse by maximizing the use of available space within a warehouse by more efficient picking and storage. There is a huge array of strategies for product allocation but those strategies focus on heuristics but maybe an RL agent can achieve similar results even being much simpler. The product allocation problem can easily be scaled down and the Logistics Support department already has a good idea about what factors make a "good" slotting, this makes it suitable for an approach that starts small and to slowly scales up from there. Slotting is also very labour-intensive and often comes down to solving a large number of small tasks such as adjusting the number of locations - also called facings - for a product. This makes the potential gains much larger. Compared to the other two problems and according to the criteria defined at the beginning of this section, the product allocation problem was picked for an implementation attempt.

5.4 Automating the slotting process

Before commencing with the technical implementation the task environment needs to be assessed. Based on the actual slotting at Albert Heijn the environment has been made and multiple scenarios, action sets and credit assignment strategies have been formulated in order to test the agent.

5.4.1 Task environment

To test whether an agent is able to solve real-world logistical challenges multiple realistic scenarios are considered. The following scenarios are tested, visualized in Figure 5.1:

Scenario A The most basic scenario with 10 pallet locations with locations on two sides of the path.

Scenario B This scenario contains 20 locations in order to assess the impact of having a DC that is double the size in terms of locations and products to slot.

Scenario C This scenario contains 42 locations and also contains a flow rack location in which slow-moving products can be slotted efficiently. Flow racks are only accessible from one side.

The layout of the task environment can also be visualized in a tabular view, ordered on the pick sequence. This is a simplified version of the layout the slotters use in their work. The tabular view can be found in Table 5.2. The last two columns are used to monitor which products were slotted initially and currently.

With the environment in place, one can consider the actions the agent can take (also denoted as S). In order to assess whether less possible actions result in faster training and better results, three types of agents will be considered. The agent types are:

Sequential The agent sequentially makes decisions. When a decision has been made for every location the agent is finished. The number of possible actions is equal to the number of products.

Partially autonomous The agent makes decisions for every product and every location. When a decision for every location has been made, the agent is finished. The number of actions are the number of products times the number of locations.

Fully autonomous The agent makes decisions for every product and every location and it decides when it is finished. The number of actions are the number of products times the number of locations plus one finished action.

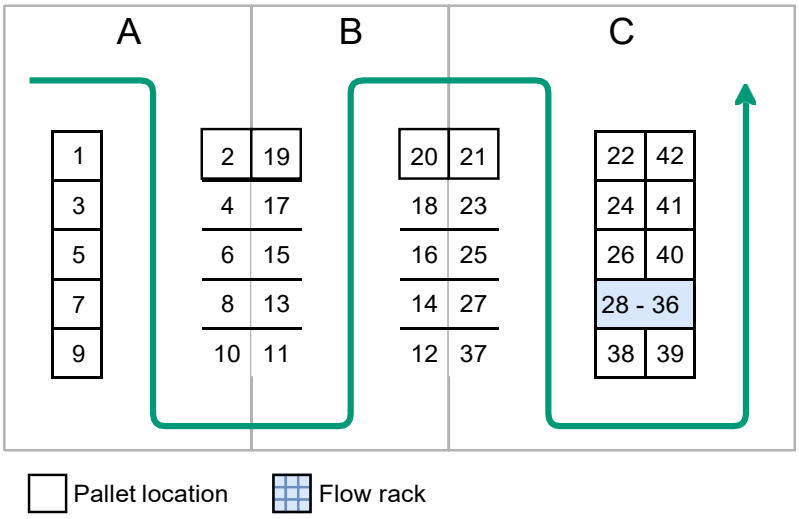


Figure 5.1: The task environment and the scenarios

Location	Next location	Flow rack	Slotted initially*	Slotted*
1	3	False	-	Coffee
2	4	False	-	-
3	5	False	-	Cookies
4	6	False	-	Soup
5	7	False	-	-
6	8	False	-	Soup
7	9	False	-	-
8	10	False	-	Rice
9	-	False	-	Toilet paper
10	-	False	-	Toilet paper

Table 5.2: An example of the locations for scenario A

For every scenario a list of products is available that need to be slotted. The product details include variables such as how many products fit in a pallet location and how many in a flow rack. The store demand forecast (SDF) of the next three

days is also taken into account so the agent can make decisions that also take into account the days to come. The products that need to be slotted are visualized in Table 5.3.

Product	Units per pallet	Units in flow rack	SDF (+1)	SDF (+2)	SDF (+3)	Stacking group	Stacking class
Cookies	220	20	1500	2200	2200	D	14000
Soup	180	12	700	500	600	C	35000
Beer	40	-	800	800	1000	A	10000
Toilet paper	24	-	250	200	300	C	40000
Rice	120	20	700	700	550	D	60000
Coffee	150	15	1000	800	900	B	12000

Table 5.3: An example of the products for scenario A

5.4.2 Reward function

Designing a reward function is one of the most difficult tasks within RL. Both primary as well as secondary goals should be taken into account when designing the function because an agent will not find the optimal policy. The reward function is the sum of rewards minus the penalties.

Rewards

The following rewards can be scored by the agent:

Product slotted When the agent should always try to slot all products, because otherwise the product cannot be delivered to the stores around the country.

Free locations Locations are limited and new products are coming to the warehouse each day, therefore it is important to keep as many free locations as possible so these can be slotted with minimal movements.

Matching SDF When the agent reserves sufficient locations for the products to be slotted it gets a higher score. When there are also enough locations for the next two days additional points are scored. This way the agent creates an slotting that can requires minimal movements at the next iteration. The number of locations is determined by the estimated store demand forecast (SDF), the location type and the maximum number of replenishments per day. For pallet locations this is set at 7 times a day while for flow racks this is only 2 times a day.

Penalties

The rewards are complemented by penalties in order to make sure that an optimal slotting also receives the optimal score. The penalties used are:

Movement Every location with a different product means a movement, which takes time and therefore should be minimized as much as possible.

Facings not adjacent Due to limitations in the WMS of Albert Heijn, facing locations can only be adjacent. A penalty is therefore imposed when the agent attempts to slot products on two separate locations.

Stacking group violation Stacking groups are important to ensure containers are stacked correctly without damaging goods. The stacking groups make sure that heavy beer crates are on the bottom of the containers whereas light products are used on top.

Stacking class violation The stacking class is a number that should be ascending when following the pick sequence to ensure products of similar dimensions are slotted near each other for easier and more stable stacking by the order pickers.

Albert Heijn recently implemented a special load carrier algorithm (LCA) to increase the load factor on its containers. Where previously the maximum weight of containers was easily reached with heavy bottles of soda, resulting in containers that were not full, the LCA now smartly distributes the weight over multiple load carriers. The stacking groups were implemented for the LCA and the slotting needs to make sure these groups are in a specific order while also take into account the stacking class. The stacking class is a number that is build up by multiple factors like weight, dimensions and whether the box is fragile or not. With a correct slotting, the resulting containers look like the one depicted in Figure 5.2. The stacking groups are "A", "B", "C" and "D". Beer crates are "A", "B" are black crates that have familiar dimensions and can easily be stacked on top of beer crates. "C" are the cover groups which are boxes that create a nice cover from which everything else - stacking group "D" - can be stacked. "A", "B" and "C" products all need to be slotted adjacent without an interruption in alphabetical order. "D" can be slotted everywhere in between as long as it does not interrupt a sequence of the other groups. An example of a correct slotting looks like: [A, A, B, B, D, D, C, D], incorrect is: [A, A, D, D, A, A, B, C] because stacking group "A" is not slotted adjacent.

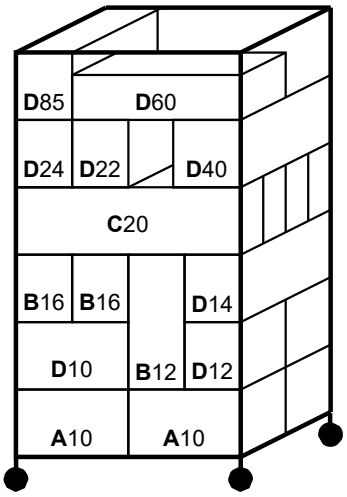


Figure 5.2: Sample container with correct stacking group and class

Verifying the reward function

To verify whether this combination of rewards and penalties yield the desired result the score is verified. For each scenario an optimal slotting was made that has no penalties, maximum rewards, maximum free locations and a limited number of movements. For scenario A, the example slotting in Table 5.2 will result in the scoreboard found in Figure 5.4.

Reward	Score	Occurrences	Total
Product slotted	+15	5	75
Free locations	+2	3	6
Matching SDF (+1)	+15	5	75
Matching SDF (+2)	+10	4	40
Matching SDF (+3)	+5	4	20
Free locations	+2	3	6
Movement	-1	7	-7
Facings not adjacent	-15	1	-15
Stacking group violation	-15	0	0
Stacking class violation	-10	0	0
			200

Table 5.4: The scoreboard for the slotting as shown in Table 5.2

Credit assignment

The credit assignment problem is one of the most difficult problems in the RL field. Primarily because it is very difficult to assess the value of an action. Because the slotting process can have both immediate as well as sparse rewards, both strategies will be used. This will also be valuable with the different types of agents, e.g. whether the fully autonomous agent will learn whether or not to decide when it is finished. The credit assignment strategies are:

Immediate The agent gets an immediate reward after every action and nothing when finished.

Sparse The agent receives no immediate rewards but only the full reward when finished.

5.4.3 Implementation

First the state is discussed and how it is passed to the agent. Finally an algorithm was selected, the agent is discussed and how its (hyper)parameters were tuned.

Passing the state to the agent

Based on the state, the agent should take actions. When for example the task is identifying handwritten digits in an image the agent is served with a low resolution grayscale multidimensional array with the intensities of white as a float between 0 and 1. This array is flattened, which just transforms this array [[0.6, 0.2, 0.3], [0.1, 0.2, 0.3]] to [0.6, 0.2, 0.3, 0.1, 0.2, 0.3]. Each value in the flattened array will be the input

layer on which the network will learn.

For the slotting we want a similar flattened array that is able to describe exactly what the current slotting looks like. For this "one hot encoding" will be used. Every product that can be slotted including an empty spot will be encoded as an array of the same length containing a single 1. This is done for every location in the warehouse, resulting in similar multidimensional array as with the digits example. An example of one hot encoding can be found in Table 5.5. When one hot encoding the following state: [bread, soup, -, -] the following multidimensional array will be created: [[1, 0, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1], [0, 0, 0, 1]]. When flattened this can be used as input for the neural network and every possible state can be described.

Product	One hot encoded
-	[0, 0, 0, 1]
Soup	[0, 0, 1, 0]
Deodorant	[0, 1, 0, 0]
Bread	[1, 0, 0, 0]

Table 5.5: One hot encoding on 3 products.

Selecting an algorithm

The first attempt for automating the slotting was made with Q-learning. This algorithm, developed by Watkins in 1989 works by assessing the quality of an action [44]. The Q-learning algorithm performed well in the small scenario that it was able to hold in memory. Q-learning is very precise but therefore not scalable beyond small task environments. The next step was to introduce neural networks and sacrifice precision for scalability.

With DQN the task environment could grow larger to around 20 locations without sacrificing precision. But there was another problem that with larger task environments after extensive training the agent started to overestimate the Q-values. This was especially difficult for the fully autonomous agents which never learned to finish the slotting because its belief of always having great alternatives. The overestimation for the other types agents did not suffer in terms of its policy, however it made validation particularly difficult. After long training the Q-values were no longer realistic and optimizations made were greatly influenced by the number of episodes the agents were trained for.

To counter the effect of overestimation of Q-values the Double DQN (DDQN) was implemented. Because DQN is chasing a moving target, the DDQN resulted in lower estimations but its results remained volatile, especially when scaling up towards larger task environments such as scenario B and C.

Beyond scenario A and B, the DDQN was not able to reach the optimal slotting with a reasonable amount of training. Therefore the Advantage Actor-Critic (A2C) algorithm was implemented. A2C outperformed DDQN in terms of training time and accuracy. Because this implementation showed the most potential, it has been used for the remainder of the implementation.

A2C agent design

The neural network structure of the A2C agent can be found in Figure 5.3. The input layer, the body and the output layer can all be adjusted to fit the environment. When working with imagery data it is possible there are one or more convolutional layers before the input layer. The neural network calculates multiple values, the Q-values for an action given state $Q(s, a_1)$ and the value of a certain state $V(s)$. The actor uses the Q-values to decide which action to take whereas the critic evaluates the current state. Combined the agent is able to calculate its advantage. If the actual reward turns out to be better or worse than expected, the weights of the network are adjusted accordingly. The agent has been implemented using Python and Tensorflow 2 and can be found in appendix D.

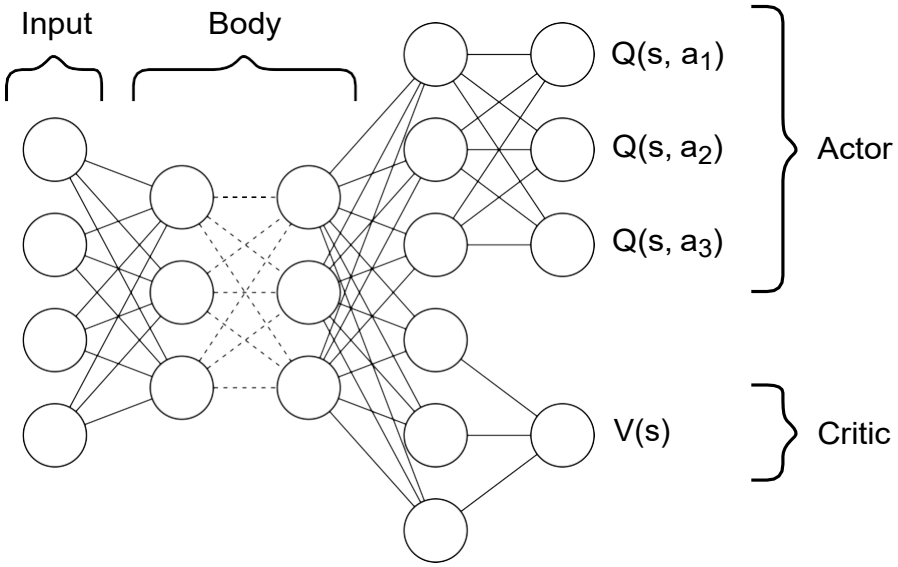


Figure 5.3: The neural network of A2C

Tuning the (hyper)parameters

Multiple (hyper)parameters can be tuned in the model the agent uses to increase its performance. For the A2C algorithm an attempt is made to tune the parameters and increase the performance compared to a baseline. In order to see the impact on performance when the task environment doubles in size and complexity all tests are ran on both scenario A and B. Every test is ran at least three times for each scenario to reduce the problem of reproducibility.

First a comparison between the different types of agents was made to see how these performed. The semi and fully autonomous agents were not always able to perform well and in some attempts the agents never completed the slotting when tested. Both agents sometimes got stuck trying to perform an action that had no effect in the environment, for example placing a product on a location that was already slotted with that particular product, this leads to a loop which takes a lot of time to get out of. The semi autonomous agent was required to make a product allocation decision for each location, because the agent has no prior knowledge it took a lot of updates to finish the same amount of episodes as the sequential agent. A similar behaviour was found for the fully autonomous agent as it has to decide when it has completed allocating products. For the analysis

the sequential agent will be used as it has proven to be most stable and has a fixed number of actions that is required to complete an episode, which eases the comparison on tuning various (hyper)parameters. For comparison, the results of the semi autonomous agent can be found in Figure 5.4, the fully autonomous agent in Figure 5.5 and for the sequential agent in Figure 5.8.

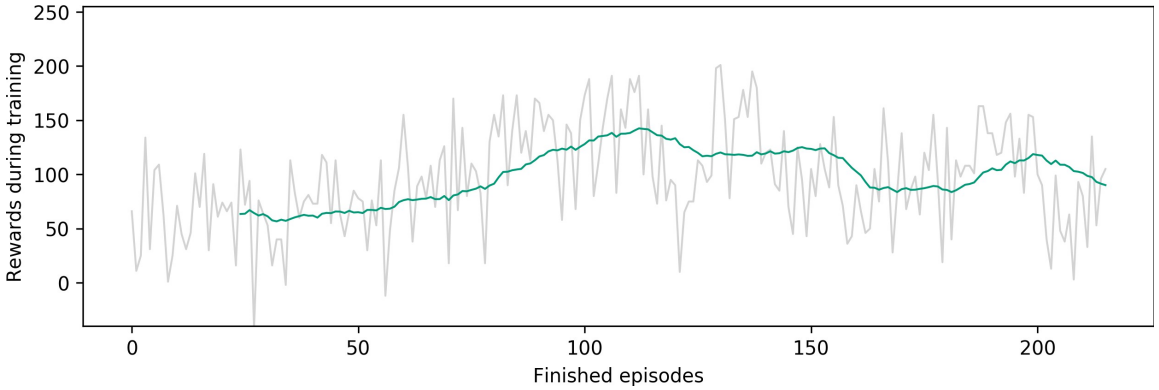


Figure 5.4: Results for semi autonomous agent for scenario A

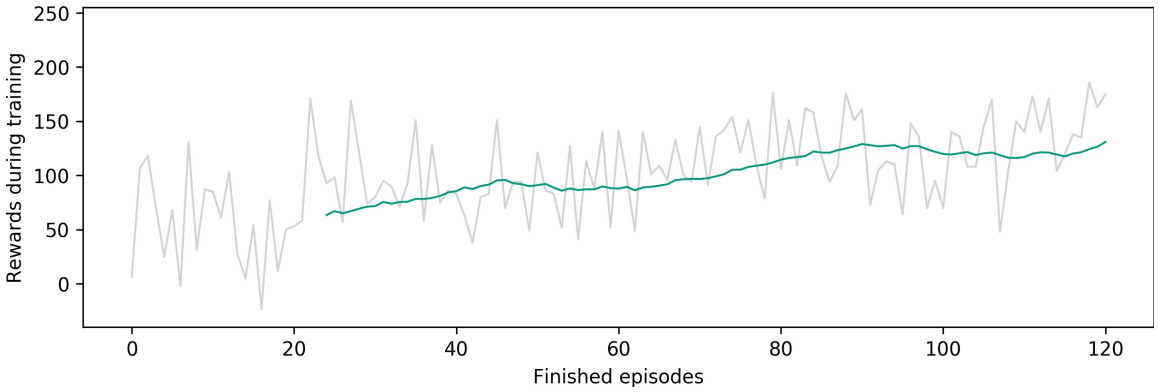


Figure 5.5: Results for fully autonomous agent for scenario A

Having immediate or sparse rewards can also have impact on the performance of the agents to be trained. In general, having immediate results is beneficial for RL because it allows the agent to learn which actions have lead to the final result instead of evaluating all preceeding actions equally. The precondition is that the reward function is able to take future actions into account. To test which reward function works best for the product allocation problem, both reward functions were tested for scenario A and B. The results are depicted in Figure 5.6 and 5.7. In scenario A, the results are clearly in favor of immediate rewards in terms of rewards however a sparse reward function seems to be more stable during training. In scenario B the results are very similar, when increasing the task environment the difference between the reward functions was not significant. For the remainder of the analysis, the immediate reward function will be used. In the following paragraphs each (hyper)parameter is discussed. The default values used in the tests can be found in Table 5.6.

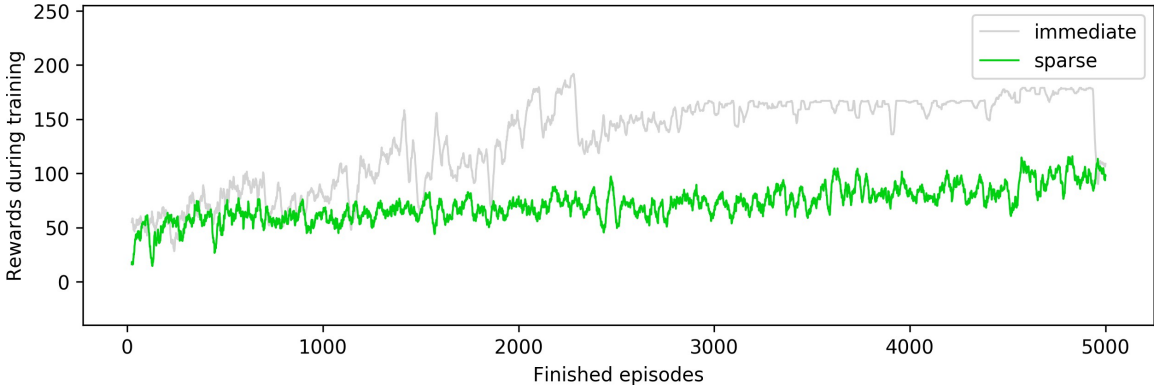


Figure 5.6: Results based on sparse and immediate rewards for scenario A

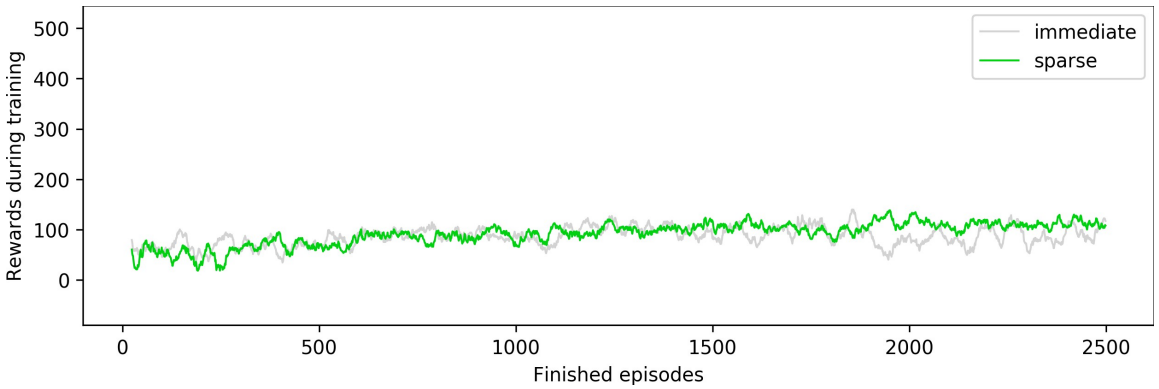


Figure 5.7: Results based on sparse and immediate rewards for scenario B

Parameter	Default value
Agent type	Sequential
Reward function	Immediate
Updates	2500
Batch size	20
Number of nodes	512
Hidden layers	2
Entropy	0.001
Gamma	0.95
Learning rate	0.0005

Table 5.6: Default parameters for the A2C algorithm

The results of the baseline tests for scenario A and B can be found in Figure 5.8 and 5.9. The baseline for scenario A took twice as long as scenario B, which makes sense because the task environment is double the size. Each graph shown in this section the Y-axis is set so that reaching the upper value means the agent performs optimally, the lowest Y-value corresponds to the starting score. For scenario A the Y-axis of (-40, 255) was used and for scenario B (-90, 545) respectively.

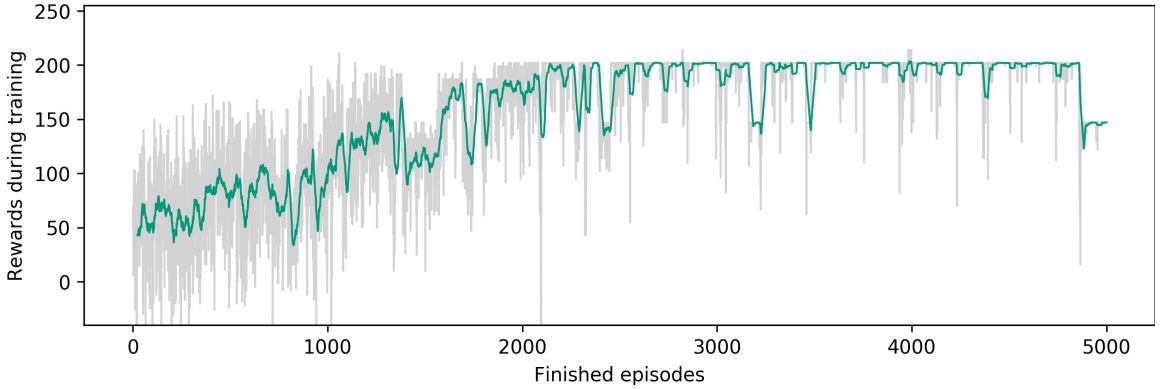


Figure 5.8: Baseline results for scenario A

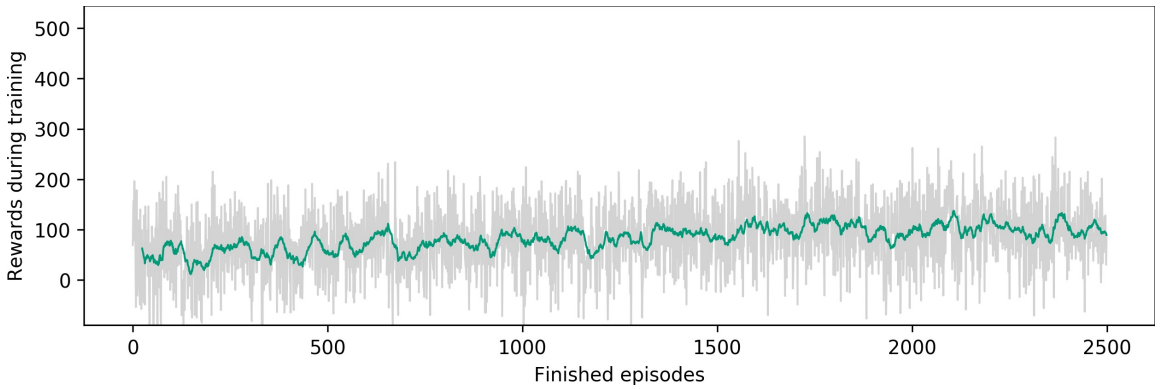


Figure 5.9: Baseline results for scenario B

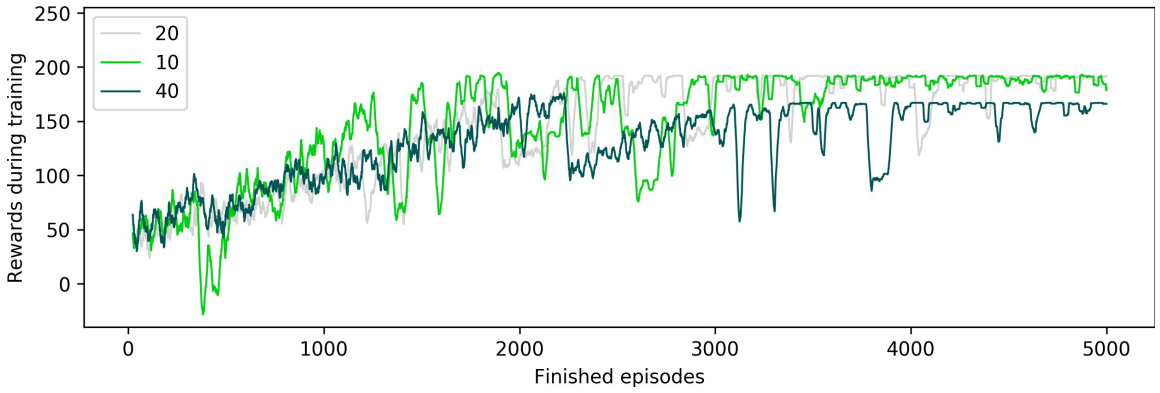


Figure 5.10: Rewards with various batch sizes for scenario A

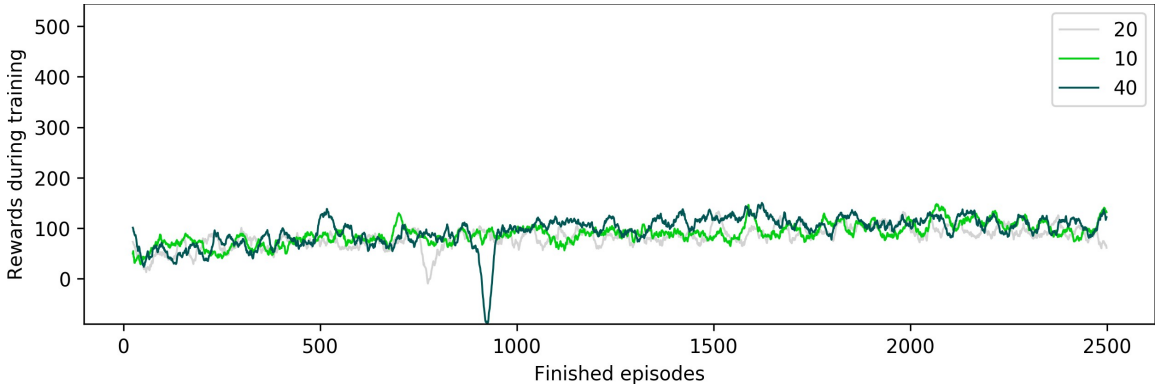


Figure 5.11: Rewards with various batch sizes for scenario B

Deep neural networks improve by feeding them more data. When reducing the batch size, the network inside the agent is updated more frequently (with a higher number of updates), increasing the load on the graphics processing unit (GPU). Depending on the GPU capacity of the machine, a lower batch size could speed up learning. The goal is to find a batch size that leads to optimal learning that the GPU is able to handle. The batch sizes tested were 10, 20 and 40. In scenario A, found in Figure 5.10, both batch size 10 and 20 perform better than the largest batch size however batch size 10 reached the local optimum faster although being more volatile in the process. In scenario B, found in Figure 5.11, the difference in batch sizes is limited. The impact of larger batch sizes in a bigger environment are limited.

Increasing the number of nodes and layers enables the agent to learn more complex problems but increases the likelihood of overfitting, making perfect decisions in a known state, but having no clue in a state seen for the first time. When the agent used 4 hidden layers the performance decreased both in scenario A and B. In scenario A having 0 or 2 layers made almost no difference in the final performance however having 2 layers looks more unstable, as can be found in Figure 5.12. The most stable learning also seems to be the case when the agent attempted to solve the product allocation problem in scenario B depicted in Figure 5.13.

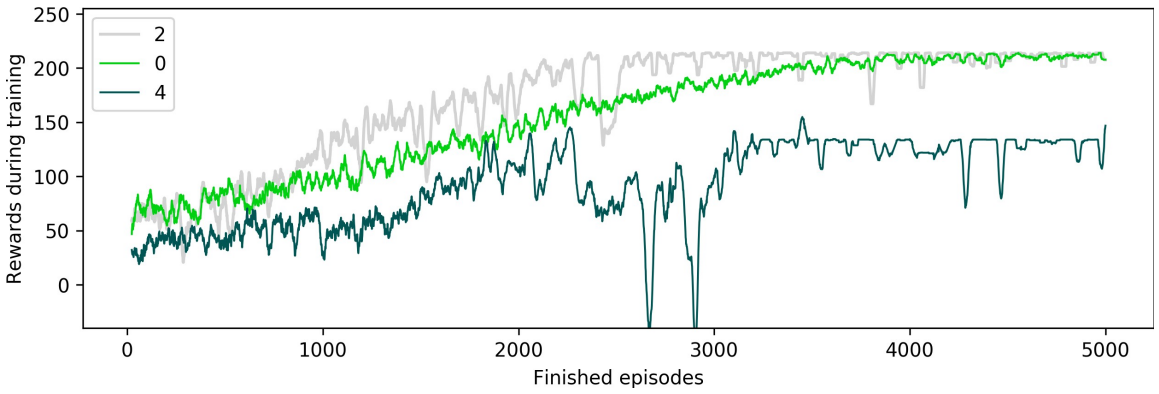


Figure 5.12: Rewards with 0, 2 and 4 hidden layers in scenario A

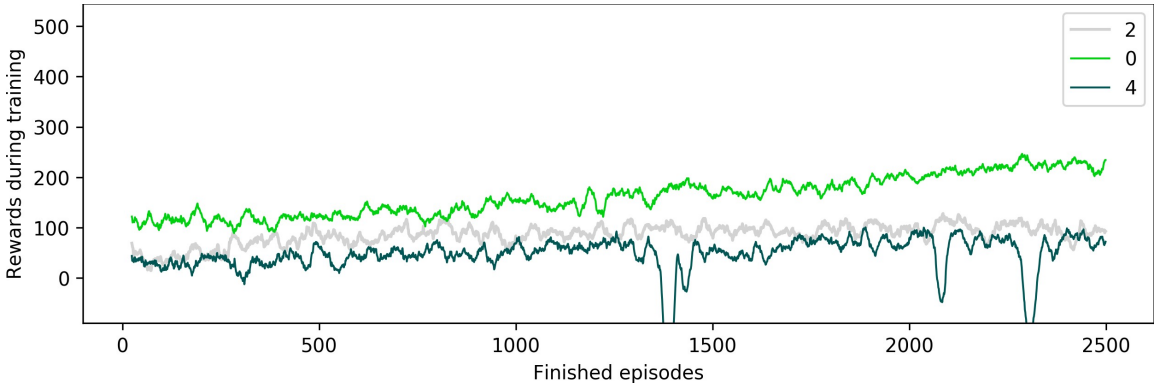


Figure 5.13: Rewards with 0, 2 and 4 hidden layers in scenario B

Apart from the number of layers, the number of nodes per layer can also be adjusted. In order to assess whether more nodes per layer increases the performance a test with 256, 512 and 1024 nodes was performed on both scenario A and B. In

the smaller scenario A, having less nodes (256) made the learning more stable compared to having more nodes, see Figure 5.14. Shown in Figure 5.15, when the scenario gets larger the differences between having 256 or 1024 are lower compared to scenario A however having 256 and 512 nodes still outperformed having 1024 nodes per layer in scenario B.

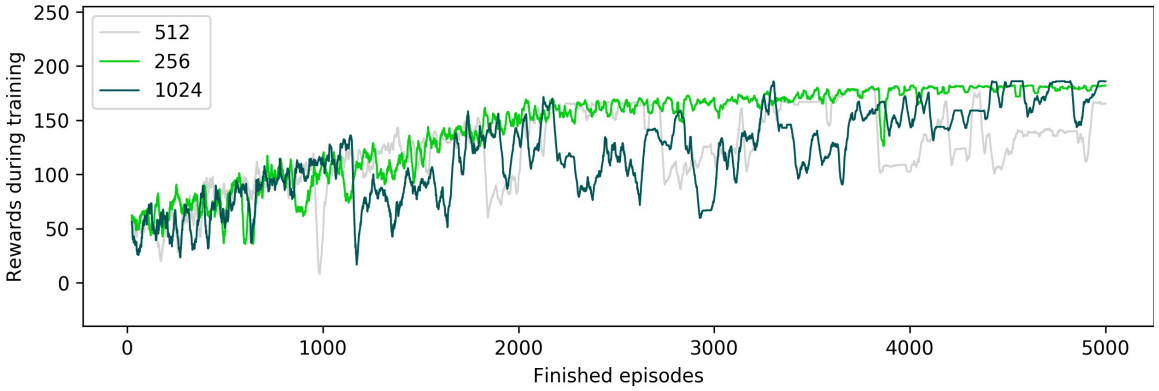


Figure 5.14: Rewards with 256, 512 and 1024 nodes per layer in scenario A

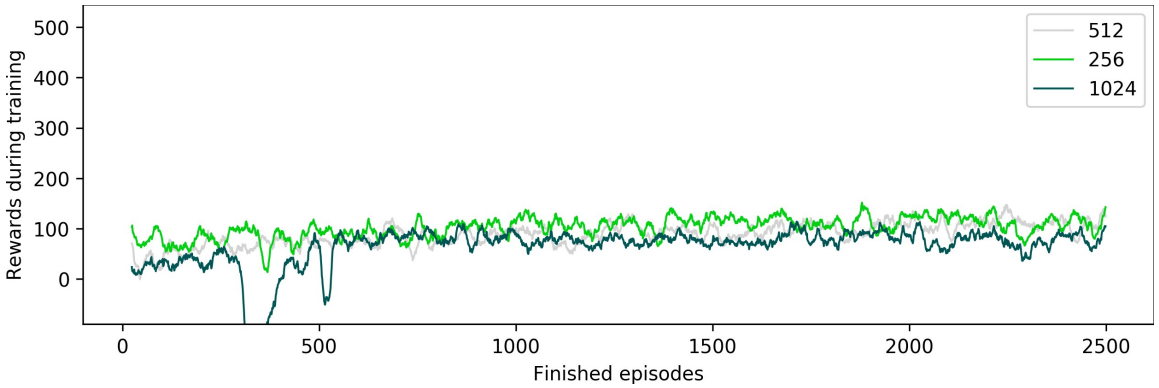


Figure 5.15: Rewards with 256, 512 and 1024 nodes per layer in scenario B

The learning rate in neural networks decide how much a the old value is overwritten by the new value. Doubling the learning rate makes the agent "learn" twice as fast. The problem with having a large learning rate is that the agent could slowly forget important older experiences. Various learning rates were tested for scenario A in Figure 5.16 and scenario B in Figure 5.17. In scenario A having a larger learning rate made the agent learn fast sometimes but after 2500 episodes its average performance did no longer improve whereas the lower learning rate agents kept on improving. In scenario B the lowest learning rate performed best overall but the agent with a high learning rate learned faster in the first 1500 episodes before dropping back down again.

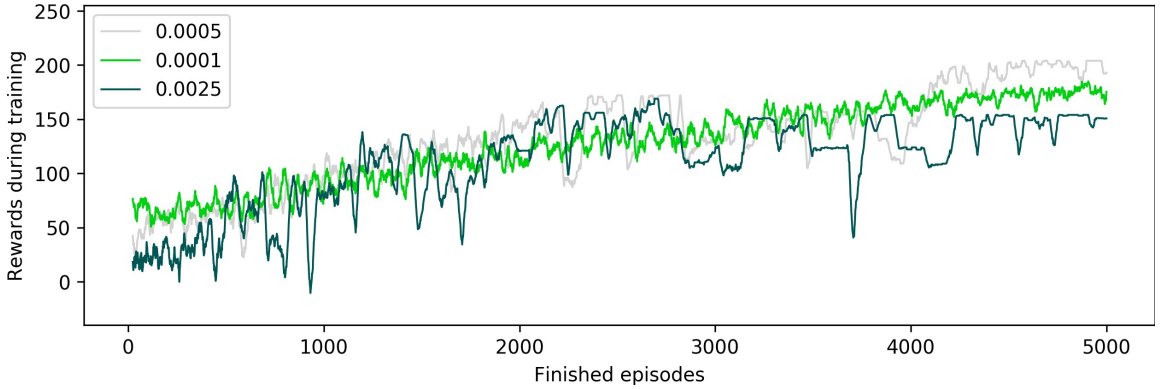


Figure 5.16: Rewards with various learning rates in scenario A

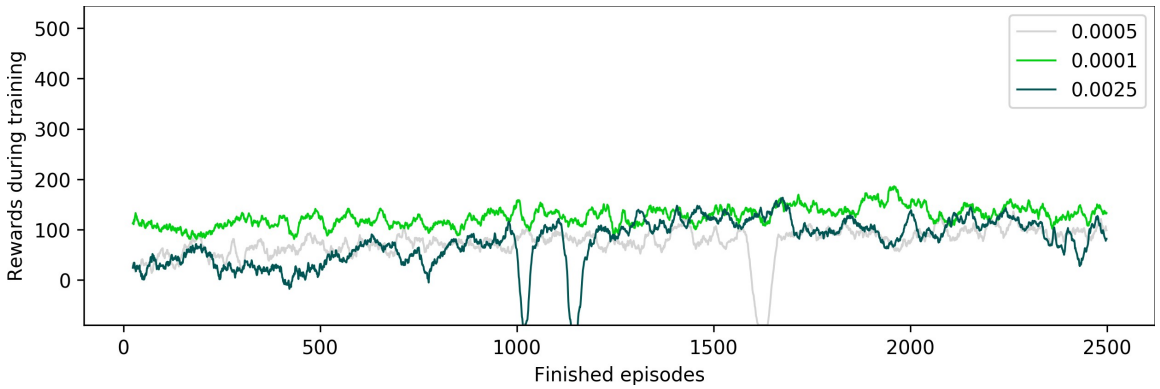


Figure 5.17: Rewards with various learning rates in scenario B

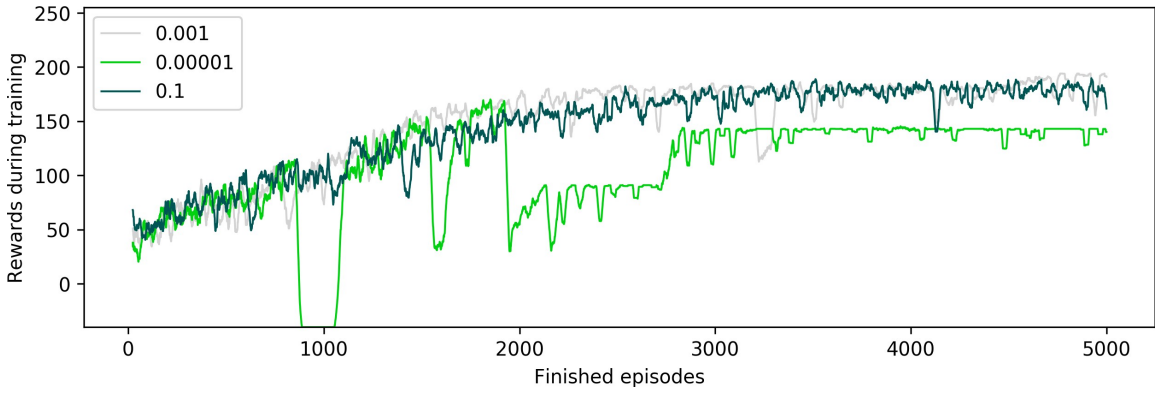


Figure 5.18: Rewards with various entropy values in scenario A

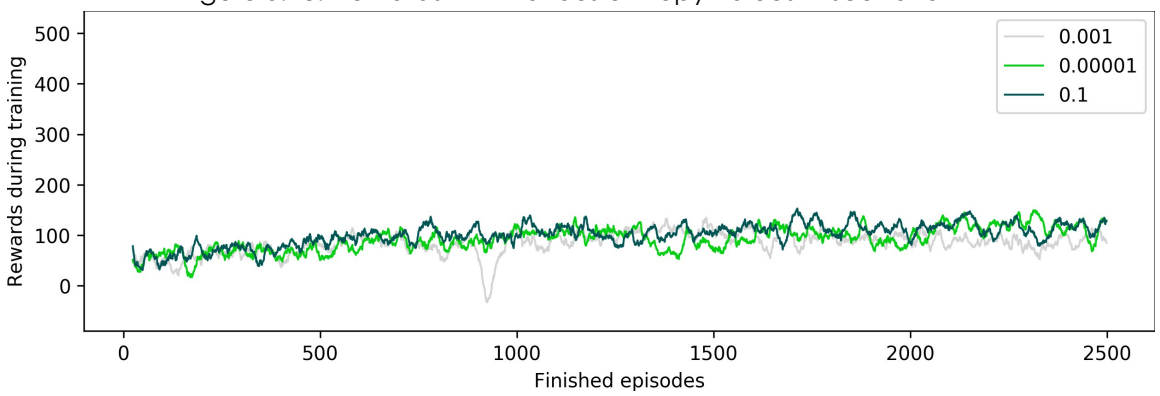


Figure 5.19: Rewards with various entropy values in scenario B

Despite the fast convergence of the agent, it failed to reach the maximum score. The exploration of the agent might need to improve, here is where the entropy comes in. Entropy is an important aspect in the A2C algorithm, because it provides a bonus for when the agent explores. This bonus encourages the agent to take actions more unpredictably, and potentially discover another better solution. A low entropy value makes the agent stubborn and take actions it took before, a high entropy makes the agent try different actions. In scenario A the default entropy made the agent learn the most stable, comparable to a high entropy of 0.1, see Figure 5.18. With the lowest entropy the agent learns in a similar fashion however there are some noticeable drops during training and after 2000 episodes the agent fails to keep up and learn something new. In scenario B, shown in Figure 5.19, the differences are almost unnoticeable, this is largely because the agents are still learning about the environment even when always picking actions the agent took before. More episodes could eventually show similar results to scenario A.

Gamma (γ) is used to discount future values. Getting a lower reward immediately could be more valuable than being uncertain a larger reward will be reached later on. This is were gamma comes in and tweaking it could lead to the agent making different decisions. For scenario A, found in Figure 5.20, a gamma of 1 resulted in the highest rewards and the rewards dropped when gamma was lower. This can be explained due to the fact that future rewards in the product allocation problem do not change, there is not another agent changing the environment that makes it important to discount future values. In a larger scenario discounting is valuable as the agent is less certain about rewards it is going to get all the way at the end of the puzzle. In scenario B, depicted in Figure 5.21, a lower gamma is beneficial. Gamma is also more important for the fully-autonomous agents as it can overwrite its own actions, without a discount factor it could slot forever without making the decision to end the process.

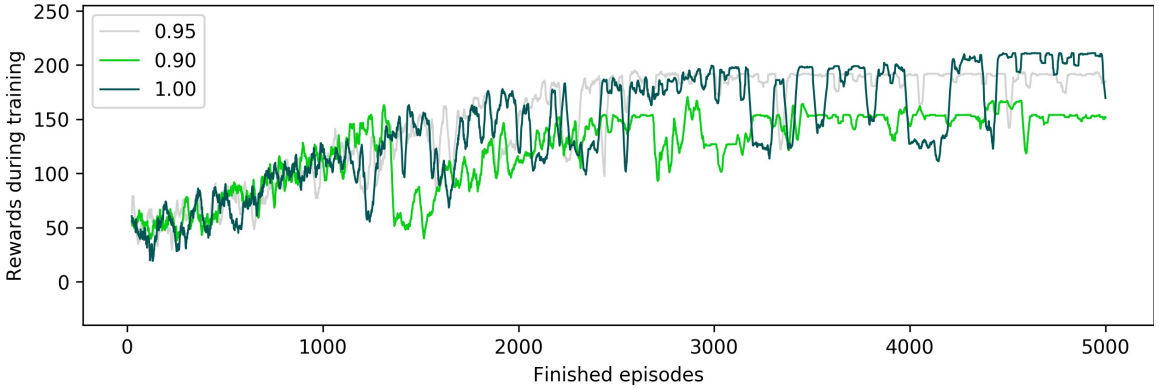


Figure 5.20: Rewards with various gamma values in scenario A

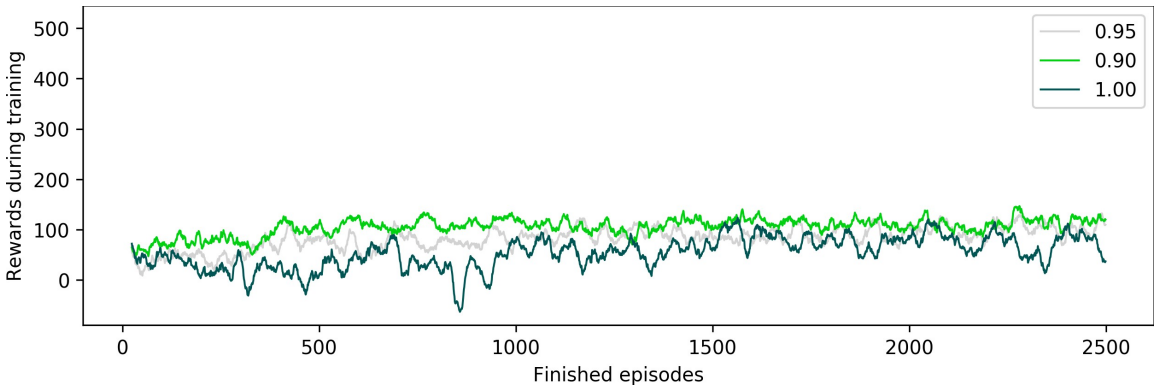


Figure 5.21: Rewards with various gamma values in scenario B

Parameter	Scenario A	Scenario B	Scenario C
Agent type	Sequential	Sequential	Sequential
Reward function	Immediate	Immediate	Immediate
Updates	10000	20000	20000
Batch size	10	20	42
Number of nodes	256	512	1024
Hidden layers	0	0	0
Entropy	0.001	0.005	0.01
Gamma	0.95	0.95	0.95
Learning rate	0.0001	0.0005	0.001

Table 5.7: Optimized (hyper)parameters used per scenario

Based on the (hyper)parameter tuning performed for each scenario the default parameters have been identified. These parameters will be used in the experiment in the next section. The parameters can be found in Table 5.7.

5.5 Intelligence amplification

So far only the performance of the agent is considered, in this section the human performance is also taken into account. Intelligence amplification is the symbiotic relationship between an human and an intelligent system. The general idea is that humans excel at creative tasks whereas artificial agents excel at computa-

tionally intensive tasks [17]. This symbiotic relationship between the human and the agent could also be applicable to reinforcement learning. The aim of this section is therefore to find out to what extent a reinforcement learning agent can support the operational staff in allocating products in the warehouses of Albert Heijn. When the agent attempts to solve the slotting puzzle on its own it learns and finds a local optimum, this is however not always the global optimum (the most desirable outcome). The slotter should be able to find the optimum, when given enough time. An experiment has been developed to identify whether a partnership that emphasizes the strength of the slotter and the agent is beneficial in terms of performance, quality being the score of the slotting and time being the time the slotter has devoted to the product allocation problem. The idea is that the slotter starts with partially allocating products to locations and then letting the agent complete the task.

5.5.1 Experiment setup

To validate whether intelligence amplification is able to improve the performance of the slotters the following experiment was created. 50% of the target population (4 participants) of slotters were asked to take part. The slotters had to solve one or multiple slotting puzzles that correspond to the scenarios considered at the beginning of this chapter. First a baseline was established by slotters that were asked to complete a randomly assigned scenario without the help of an agent. Other slotters then got the same scenario however this time they were able to use an agent. Finally the agents ran the scenarios without the slotters. Time spent by the agents is not taken into account as this is hugely dependent on the available hardware and the costs of hardware is beyond the scope of this research.

Validation research goal

The following knowledge questions have been identified to find out whether or not intelligence amplification increases the performance of slotters:

- How does the slotter with the agent compare to the baseline in terms of time and quality? (Trade-off question)
- What effects are produced by the interaction between the slotter and the agent? (Effect question)
- What happens when the problem context becomes bigger? (Sensitivity question)

The first question is essential as it gives an idea about whether the agent can achieve similar quality in less time than the the slotter. The second question focuses on the partnership between the agent and the slotter and the relationship between the two. The final research goal question is whether the intelligence amplification scales when the environment gets bigger. Combined these questions give a good idea about whether or not intelligence amplification with reinforcement learning can be beneficial and to what extent.

Samples

The slotters that got the baseline experiment were asked to solve a small slotting puzzle similar to their daily work although being much smaller. The scenarios used

can be found in appendix C. The slotters are asked to solve these small puzzles by hand starting with an empty warehouse. The slotters first get an explanation of the goal of the task which is to solve the puzzle as good as possible in the least amount of time. An small example is presented to the slotters before commencing to make sure they are prepared and understand the task. With the example completed the actual experiment can start. The experiment is carried out on paper to make solving it easy and understandable. The actual puzzle to solve is on the back of the paper and when turned the experiment starts. When the slotter is finished with the task the paper is turned on its back again and the start and end time is noted. The quality of the slotting will be assessed by the rewards and penalties used throughout this thesis. The rewards and penalties are discussed with the slotter beforehand to make sure they know where to pay attention to when solving the puzzle. The participants are also asked to solve one ore more different scenarios with the agent used for a comparison later. The slotter with the agent does not have to finish the puzzle, as it has an agent to do that. An example of the experiment is discussed in the next section.

Example experiment

Because there were multiple experiments for each scenario, a small example is presented here. The participant first gets a schematic overview of the DC. In Figure 5.22 a schematic overview of a DC with 6 locations is depicted.

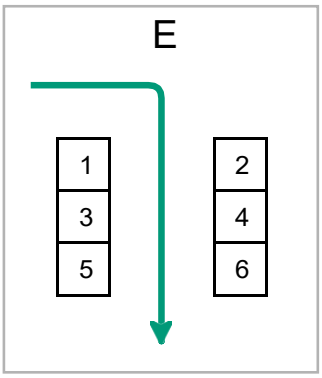


Figure 5.22: The circuit for scenario E

In Table 5.8 the products are listed that need to be allocated to one or multiple locations. The store demand forecast (SDF) for the coming three days as well as the stacking group and stacking class are listed. The maximum number of replenishments for a pallet location is 7 times a day and for a flow rack this is 2. This can be used to calculate how many locations need to be allocated for each product.

Product	Units per pallet	Units in flow rack	SDF (+1)	SDF (+2)	SDF (+3)	Stacking group	Stacking class	Lock for agent?
Toilet paper	24	-	120	140	160	D	60000	■
Soda	40	-	400	800	700	B	11000	**D**
Soup	180	12	1000	2500	2500	C	25000	**D**
Beer	40	-	800	800	1000	A	10000	■

Table 5.8: Products to slot in scenario E

Depending on the experiment the participant has one of the following assignments:

1. Allocate the product as optimally as possible while using as little time as possible.
2. Allocate the product as optimally as possible while using as little time as possible. But this time the participants do not have to finish the entire puzzle. After partially solving the puzzle, they can lock products and locations they know are allocated correctly. The agent takes over and finishes the puzzle on its own.

Location	Flow rack	Next location	Slotted initially	Optimal slotting	Lock for agent?
1	False	3	-	Beer	■
2	False	4	-	Soda	**D**
3	False	5	-	Beer	**D**
4	False	6	-	Soda	■
5	False	7	-	Beer	**D**
6	False	8	-	Soda	**D**

Table 5.9: Locations and optimal slotting for scenario E

5.5.2 Results

The results of the experiment are presented in Table 5.10. The results of the agent on scenario A, B and C can be found in Figure 5.23, 5.24 and 5.25 respectively. The (hyper)parameter tuning really paid off as in all scenarios the learning is very consistent and all agents converge towards a local optimum. Without the agent, the participants are able to get the maximum reward however when supported by an agent the time could be reduced without the quality of the slotting reducing substantially. By locking both locations and products the actions required to be considered by the agent dropped significantly, making it more likely to find a local optimum closer to the maximum reward.

5.6 Conclusion

The department of Logistics Support is AI novice and numerous activities have resulted in more knowledge about AI and the impact it could have on the department. Multiple business processes have been assessed and together with the department the slotting has been identified as a process that is most labour-intensive and would benefit from an agent supporting the slotter in the daily operations. When trying multiple algorithms the A2C achieved the highest sample efficiency and performance. The (hyper)parameters of the algorithm were tested on two scenarios to get an initial idea about their impact when scaling up towards

Participant	Actor	Scenario	Employee time	Employee	Δ	Agent	Δ
-	Agent	A	-	-	-	**202**	+242
Participant 1	Employee	A	5 minutes	**230**	+270	-	-
Participant 3	Employee	A	11 minutes	**255**	+295	-	-
Participant 4	Employee	A	10 minutes	**255**	+295	-	-
Participant 2	Combination	A	7 minutes	60	+100	**235**	+175
-	Agent	B	-	-	-	**438**	+528
Participant 2	Employee	B	16 minutes	**545**	+355	-	-
Participant 3	Combination	B	7 minutes	30	+120	**488**	+458
Participant 1	Combination	B	7 minutes	245	+355	**502**	+257
-	Agent	C	-	-	-	**729**	+935
Participant 2	Employee	C	34 minutes	**1193**	+1399	-	-
Participant 4	Combination	C	7 minutes	-60	+146	**822**	+882

Table 5.10: Performance of the participants for different actors and scenarios

the puzzle the agent has to solve in a real-world scenario. The experiment aimed at intelligence amplification resulted in a higher overall performance for slotters using the agent.

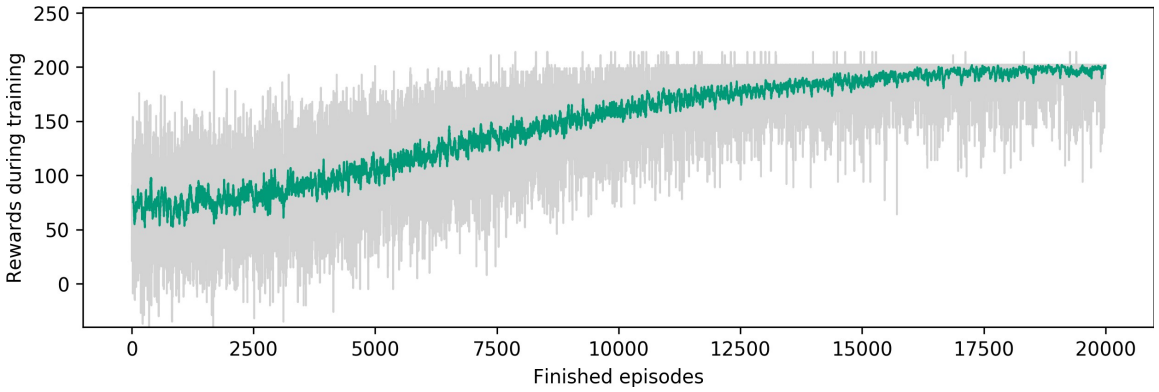


Figure 5.23: Rewards of the agent with optimized parameters in scenario A

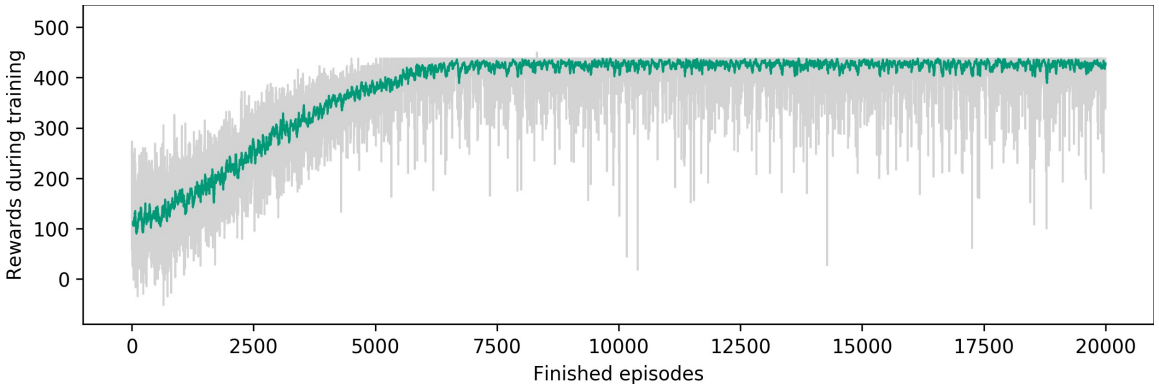


Figure 5.24: Rewards of the agent with optimized parameters in scenario B

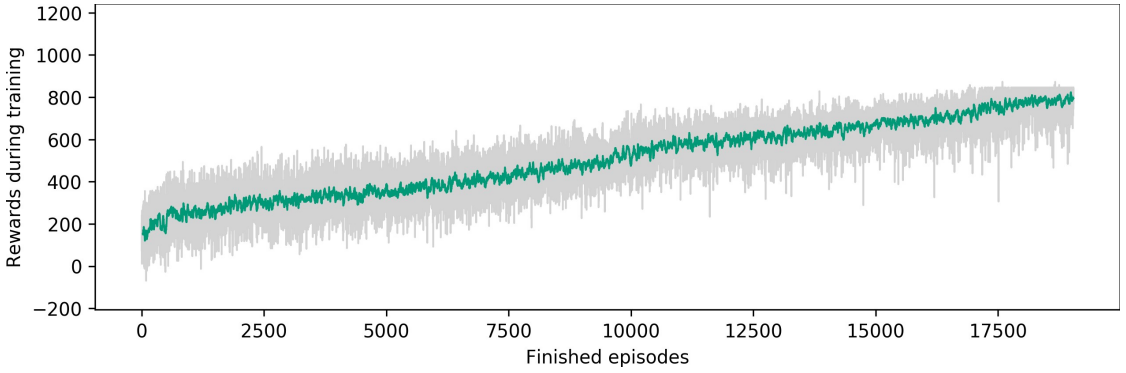
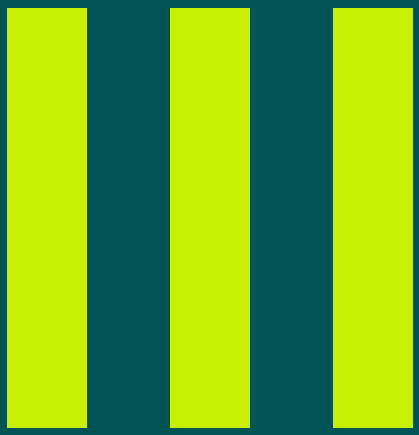


Figure 5.25: Rewards of the agent with optimized parameters in scenario C



Treatment design

6. Requirements specification

In order to transfer the experience and knowledge of the exploratory research, requirements have been identified for the model. Defining requirements helps in deriving useful guidelines for possible treatments [46]. In this chapter the requirements for the model are identified as properties desired by stakeholders. The stakeholders are identified in section 6.1. The requirements are based on contribution arguments which are a result of design choices made on behalf of the stakeholders. The goal of the model is to enable logistic organizations to effectively implement reinforcement learning.

6.1 Stakeholders

Based on the exploratory research and the parties involved in implementing reinforcement learning the following stakeholders have been identified. The stakeholders have been generalized to enable other logistic organizations in mapping slightly different positions onto these. The stakeholders and their goals are:

Team lead **The member that is responsible for the operational staff and continuously improving the efficiency and quality of the operational staff.**

Operational staff **The employees responsible for carrying out the daily logistic operations, such as slotting, order picking and planning.**

Developers **The developers are responsible for developing and maintaining the software to support the operational staff. These developers do not necessarily have in-depth machine learning knowledge.**

6.2 Requirements

The requirements of the model are properties desired by some stakeholder, who committed resources and time to realize the property [46]. The requirements are split into functional and non-functional requirements.

6.2.1 Functional requirements

Functional requirements are requirements for desired functions of the model. A function is a terminating part of the interaction between an artifact and its context that contributes to to a service to a stakeholder [46]. The following functional requirements have been identified:

1. The model enables team leads to identify business processes suitable for reinforcement learning.
2. Using the model, the team lead is able to design a task environment that resembles the real-world together with the operational staff.
3. The model gives the team lead a good idea about whether or not the RL agent is going to succeed during implementation.
4. The model gives team leads an idea about the expected workload of the stakeholders during the implementation.
5. The model helps developers to tune the (hyper)parameters to increase the performance of the agents.
6. The model is can easily be adapted and tuned by logistic organizations.
7. The model is compatible with an ever expanding number of RL algorithms.

6.2.2 Non-functional requirements

The non-functional requirement for the model is that the model should be easy to use and learn for both team leads and developers. For this non-functional requirement the indicator is the effort required.

6.3 Positioning of the artifact

The model should be positioned to help logistic organizations that are just getting started with reinforcement learning or even AI in general. The model encompasses all of the technology, organization and the logistic environment and is targeted at AI novice organizations [16, 31]. The positioning can be found in Figure 6.1.

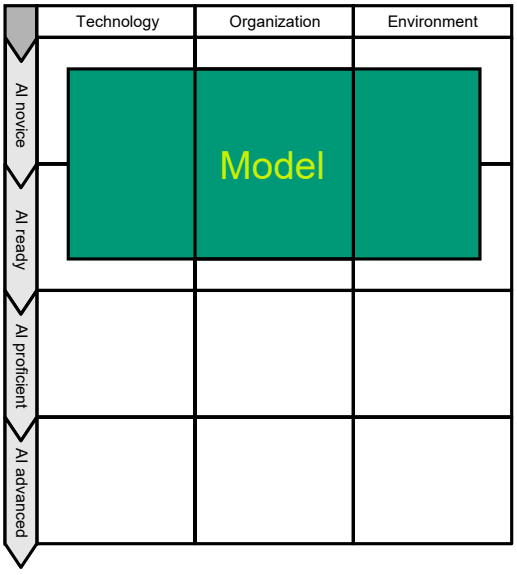


Figure 6.1: The positioning of the model.

7. Model

In this chapter the proposed model as well as its tasks for each stakeholder are discussed in detail. The model consists of recommendations that are a result of the exploratory research and the problem investigation. An overview of the model can be found in Figure 7.2. To enable other logistic organizations to adapt the model into their business processes BPMN is used, as it is the leading standard in business process modelling. The model consists of three phases, exploration, scaling up and implementation. In the following sections each stakeholder is discussed starting with the team lead.

7.1 Team lead

The team lead is the most important actor in the implementation process, as he or she is responsible for identifying suitable business processes for RL. Together with the the other stakeholders the team lead coordinates the process and decides whether or not to continue with the implementation. The expected workload for each phase is presented in Figure 7.1.

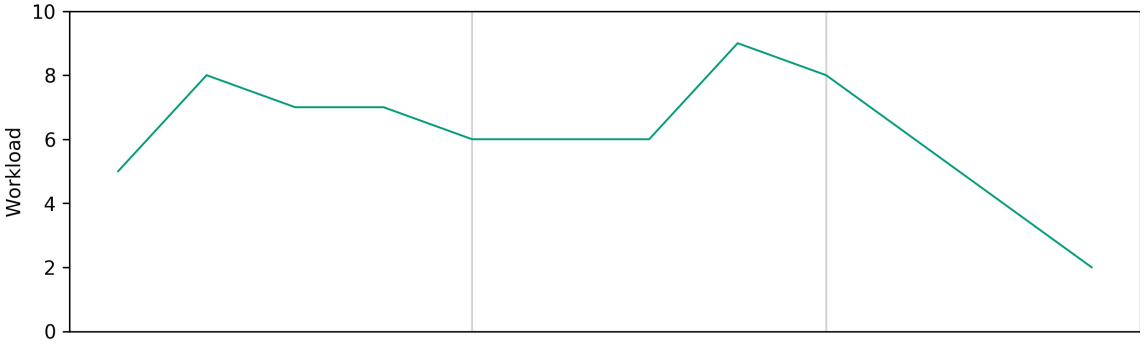


Figure 7.1: Workload for team lead

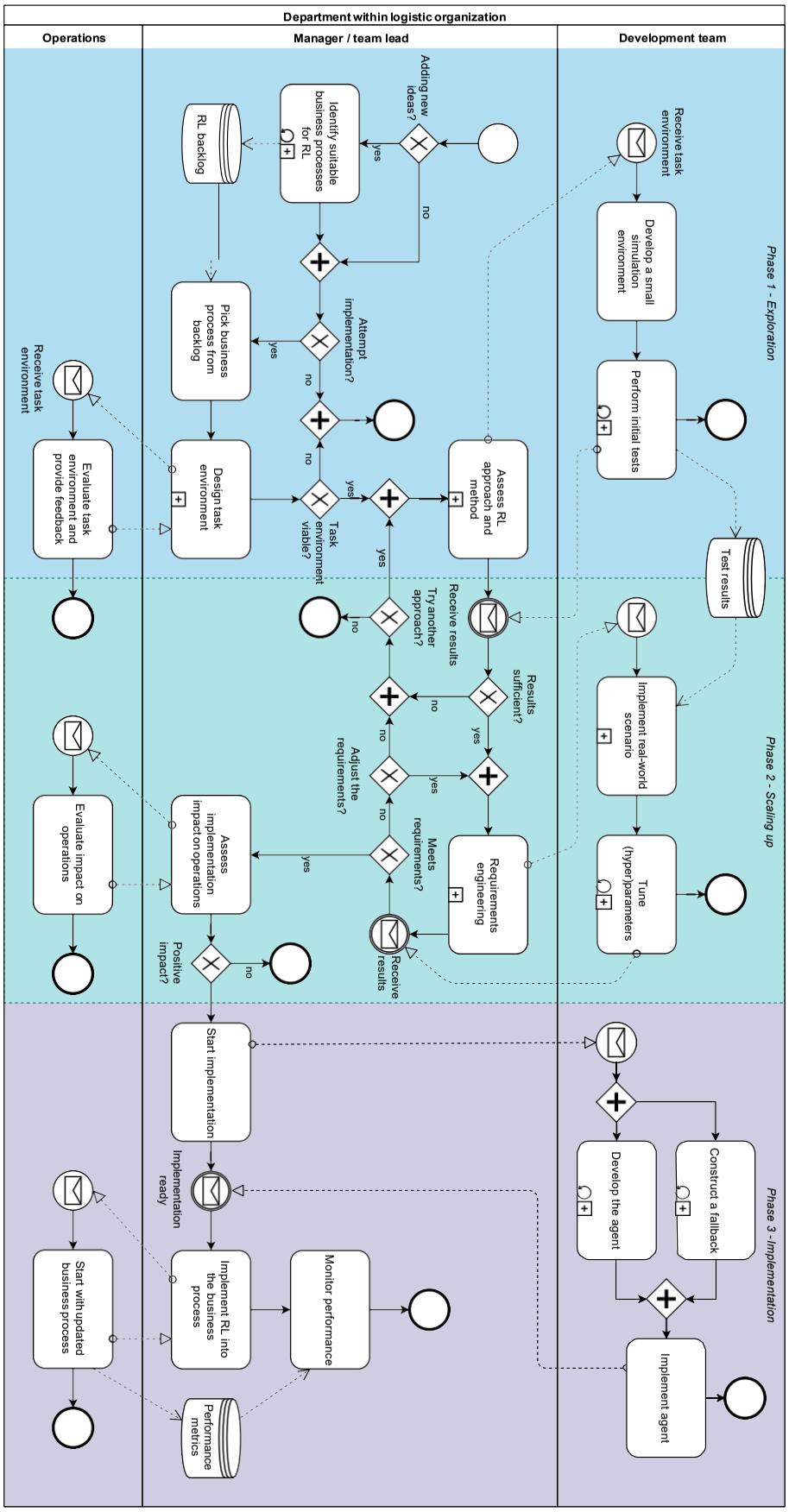


Figure 7.2: A method for RL-driven business process re-engineering

7.1.1 Identify suitable business processes

Not every task is suitable for reinforcement learning. The first step when identifying a suitable business process is defining (a simplified version of) the task environment. The task environment consists of the agent, what sensors it has and which actions it can take as well as the environment the agent is operating in. When the task environment is considered to be suited for RL by the team lead, it is added to a backlog where they are prioritized in terms of impact and expected implementation difficulty. Based on literature and the exploratory research each of the following characteristics help in identifying whether or not the task environment suits reinforcement learning:

- Fully observable or partially observable? Because RL is not very sample efficient, it is important that the agent has access to enough input data. A fully observable environment is therefore preferred over environments with limited observability.
- Single agent or multiagent? The complexity of task environments with multiple agents increases significantly. Having an agent solve a smaller part of the task environment could alleviate this problem.
- Deterministic or stochastic? Knowing exactly where the agent ends up taking a particular action helps the agent train the correct policy however when the environment is stochastic it could make it more robust to overfitting.
- Sparse or immediate results? Whether it is possible to give the agent a reward immediately after taking an action or if it only comes at the end. Immediate rewards result in faster training agents and is preferred especially when using temporal-difference learning algorithms. The shorter the delay between action and consequence, the faster the feedback loop gets closed and the easier it is for an agent to figure out a path with high rewards.
- Static or dynamic? When the environment changes when the agent is still considering an action the environment is dynamic. Comparable to stochastic environments this is more difficult for the agent to learn.
- Discrete or continuous? RL agents have been successful in both discrete as well as continuous task environments. It is important to consider the possible scenario the agent can run into.

Considering all of these task environment characteristics, the team lead can rank these business processes accordingly and decide whether or not to add it to the backlog.

7.1.2 Design task environment

The team lead makes a decision to pick a certain business process from the backlog. In this part the design of the task environment is developed. Similar to the exploratory research, the team lead specifies the details of an environment and the agent interacting with it. This is split up into three parts, the environment, the reward function and the agent types. After these activities are performed, the team lead makes the decision to ask the development team to develop a small simulation environment.

The environment

The main task is to design an environment that simulates (a simplified version of) the real-world business process. For the exploratory research this was a table consisting of all locations and the products allocated to those locations. A task environment can easily consists of multiple inputs such as tabular and imagery data.

Finally the environment should be passed to the agent in a way it can understand. In the exploratory research one hot encoding was used to pass the state to the agent. But using imagery data is also a possibility. In literature passing an image to the agent often consists of using the greyscale values of of a cropped image.

Actions

There can be multiple ways to solve the same puzzle. In the exploratory research the agent could solve the puzzle by only deciding between products and sequentially solve the puzzle however the fully autonomous agent was able to slot products on a specific location as well as decide when it was finished. With the environment in mind the team lead could come up with different ways the agent can interact with the environment. Based on the results of the exploratory research, the idea is to get a minimal set of actions of which the agent is able to reach its goal.

Reward function

Creating a reward function is not difficult however designing a reward function that encourages intended behaviour is, especially considering the agent has to be able to learn it. For the slotting multiple rewards and penalties were specified to mimic how a slotter would evaluate the product allocation. The problem however was that only slotting was evaluated using the products that were already allocated, and not on the products that still needed to be allocated. The result was an agent that started slotting certain products, slotted correctly from there however in the end did not find the global optimum. The team lead should attempt to design a reward function that gives an accurate reward in a particular state taking into account the actions the agent can take. For example when an agent can undo its action it could still learn the optimal policy.

7.1.3 Assess RL approach and method

With a task environment in place the team lead decides whether or not it is sufficient enough for the developers to develop a small simulation. This simulation is used to identify whether or not the agent is able to learn the correct policy in a small task environment before scaling up towards a more realistic scenario. In order to pick an RL approach the team lead can decide between three RL approaches which are model-based, value-based and policy gradient. These approaches are not mutually exclusive, A2C for example uses both value-based as well as policy gradient methods. Based on the task environment, the following characteristics of the task environment can ease the decision making:

1. Model-based, when the model of an environment is known.
2. Value-based, such as Q-learning and DQN, learn by estimating how good it is to take a particular action.
3. Policy gradient, deriving a policy directly.

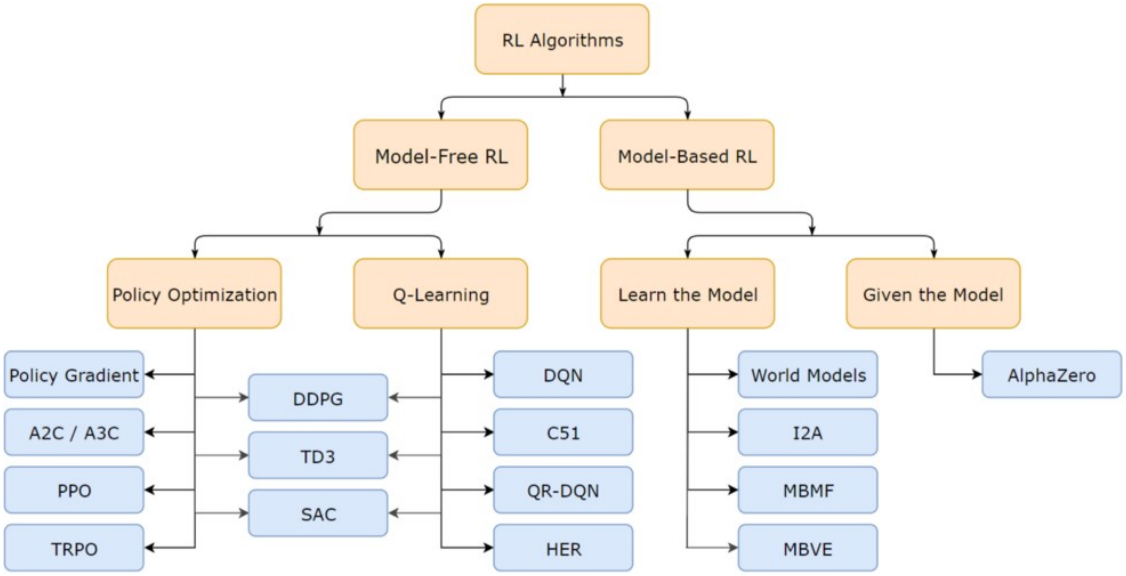


Figure 7.3: Overview of RL algorithms

OpenAI provides a clear overview of the available RL algorithms organizations could use[1]. Figure 7.3 shows an overview of the currently available RL algorithms.

Within AI novice and AI ready logistic organizations it is likely that there is limited knowledge about machine learning and how to get started. To tackle this problem the backbone of the A2C agent used in the exploratory research is provided in Appendix D to enable these organizations to get started nevertheless.

7.1.4 Requirements engineering

When the results of the first experiments show that the agent is able to learn and solve the problem presented so far, the team lead can start with the requirements engineering process. Based on what the agent was able to learn in the initial tests. These requirements should tackle key performance indicators such as the required performance of the agent and whether or not a fallback should be in place.

7.2 Development team

The developers are responsible for realizing the agent and the simulation according to the specifications designed by the team lead and the operational staff. Starting with a small proof of concept in the exploration phase the team slowly gets more involved and thus the workload increases. In the final phase the team realizes the agent that is capable of automating a business process. The expected workload for the developers is depicted in Figure 7.4.

[1] https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

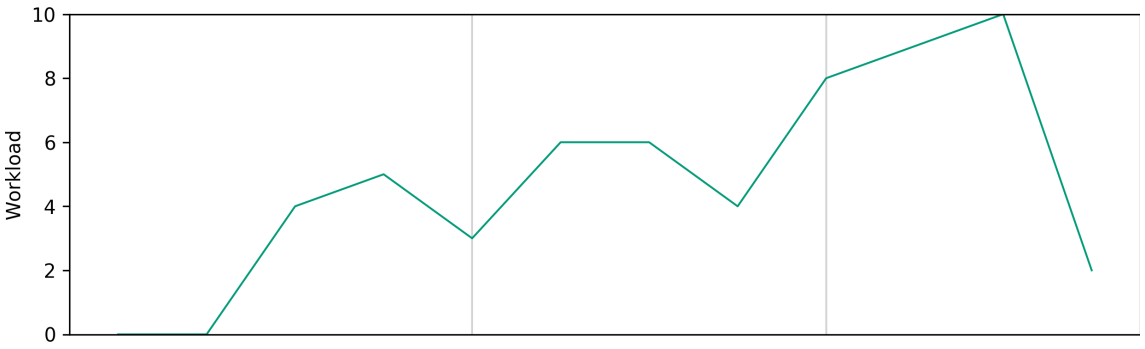


Figure 7.4: Workload for the development team

7.2.1 Develop and test a small simulation environment

The task environment, reward function and the type of RL agent thought out by the team lead are the starting point for the development team. The first goal of the team is to develop a small simulation environment and perform some initial tests. The goal of these initial tests is to find out whether or not the idea works in an environment in which the agent should be able to perform well, with initial (hyper)parameters. In the slotting case a small simulation was developed of a warehouse layout with only four locations and four products to slot. This simplified example showed the potential of Q-learning for the problem and with these initial results the team lead could continue.

7.2.2 Implement a real-world scenario

When the team lead has developed requirements and has an idea for a real-world scenario, the development team gets involved again. The team creates a real-world scenario that is a subset of the actual problem to solve. This step is important because as the task environment and reward function get more specific, it could make learning more difficult for the agent. The real-world scenario for the slotting agent were three scenarios of various sizes that take into account the store demand forecast, the stacking groups and classes as well as facings and different types of locations. Although being smaller, this can be considered a part of the puzzle the operational staff solves each and every day. With a real-world scenario in place, the developers can test the performance of the agent by tuning the (hyper)parameters. Logistic organizations that can be considered AI novice, might not have developers with affinity with machine learning. To enable AI novice organizations to start experimenting with reinforcement learning Appendix D provides the Python implementation of an A2C agent the developers can use.

7.2.3 Tuning the (hyper)parameters

When tuning the hyperparameters of any RL agent, it is important that the developers know which parameters to adjust in order to increase the performance of the agent. Much like in supervised learning, tuning the parameters is merely an activity of trail and error. Based on the tuning that was conducted on the developed slotting agent, the following remarks could be helpful when tuning the parameters:

- Number of updates and batch size, a low batch size makes the agents

network being updated more often, increasing the load on the GPU. When having a bigger environment the batch size differences are not as significant. As with almost every deep neural network, more updates are beneficial for the performance of the agent. Different from supervised and unsupervised learning, RL tends to find a local optimum that is difficult to escape from, even with more data.

- Increasing the number of nodes or layers increases the capacity of the agent to learn complex problems. It could however lead to overfitting in which the agent performs well in known states but fails when encountering states it has not seen before. In scenario A the agent with 256 nodes outperformed having 512 and 1024 nodes. The difference was less significant when scaling up towards larger environments which substantiates the idea that larger environments need more nodes as well as layers.
- Gamma is used to discount potential future rewards. This is particularly important for stochastic task environments but in the case of the product allocation the environment was deterministic.
- The learning rate is a way to control how fast an agent learns, partially overwriting old values. A higher learning rate means an agent learns faster but also focuses more on recent experiences as old ones are removed faster. As the environments got larger, the learning rate was increased to encourage the agent to identify potential success paths faster.
- Entropy or rate of exploration/exploitation is important as it controls how much the agent explores the environment or takes actions based on knowledge. Ideally starting with an agent that only explores is a good way to start, slowly lowering the entropy from there to find an optimum. In a larger environment the agent is more likely to pick an action it took before and therefore the entropy is increased accordingly.

The importance of tuning the parameters became clear in the exploratory research, after tuning the agents trained more stable and efficiently. The team lead should not rush judging the performance of the agent before proper (hyper)parameter tuning is performed as it could have a profound impact.

7.2.4 Implementing the agent (and fallback)

When the team lead has made the decision the start implementing the final agent, the workload shifts towards the developers. The development of the final agent consists mainly of scaling up the real-world scenario towards the actual business process. A fallback should also be considered as the results of RL agents are not always reproducible, meaning that it could fail sometimes. Depending on the impact of an agent occasionally not performing, the fallback should be considered. An example at Albert Heijn is the current tool that is able to make product allocation decisions based on a search algorithm. The team lead makes sure to monitor the performance of the agent after implementation and trigger the fallback if needed.

7.3 Operations

The operational staff is responsible for carrying out the business process and can be considered the end user. In each phase the operational staff is conducted multiple

times for their operational experience. Getting the operational staff involved from the start helps in developing a positive attitude towards the agent and increases the potential to adopt it [33]. The workload for the operational staff can be found in Figure 7.5.

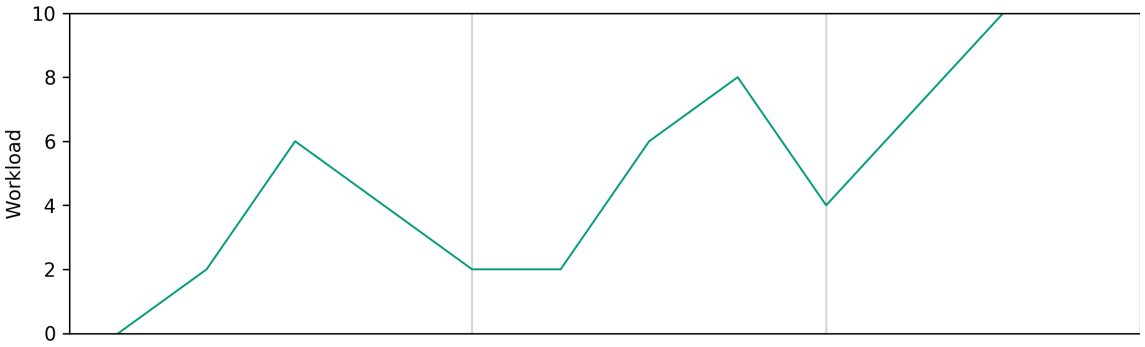


Figure 7.5: Workload for the operational employees

7.3.1 Evaluating the task environment

Together with the team lead the task environment is developed and evaluated by the operational staff. The team lead attempts to create a task environment that is suitable for RL whereas the operational staff evaluates to what extent the task environment compares to the real-world. The operational staff can identify potential problems with a reward function when comparing their own indicators for success.

In the product allocation problem a small task environment was made with a small warehouse of only 4 locations, even though this is obviously not to scale, the essence of the problem is the same. An agent attempts to - based on various inputs - allocate products in a way that is optimal. The goal is therefore to find a task environment that closely resembles the real-world puzzle, but in a simplified form. When executed correctly, the developers are presented with a task environment that is easy to implement while giving a valuable sneak peak of its potential. As shown in the exploratory research, performance could still increase even though the agent is not superior to the human by the application of intelligence amplification. If for example the final 10% of the business process is quite straightforward an RL agent could still significantly improve the performance of the operational staff. During the evaluation these potential cooperation should also be considered.

7.3.2 Evaluating the impact on the operation

Before the impact on the operations can be assessed, first the test results and the performance of the agent have to be assessed. In order to have an objective discussion the focus should be on actual results of the agent rather than speculation. In the example of the slotting agent, the agent is not always able to find the global optimum, but is able to allocate products correctly to a certain extent. The product allocation has a profound impact on the productivity of the warehouses of Albert Heijn, therefore the savings in terms of time spent by the slotters does not outweigh the loss in productivity in the warehouses.

7.3.3 Start with updated business process

With the agent implementation complete, the operational staff starts working with the updated business process. When the agent does not perform due to certain circumstances the fallback will be activated. The operational staff will be thought how to identify potential failures of the agent and how to instantiate the fallback if necessary.

IV Treatment validation

8. Model validation

The requirements of the model are specified which enable validation by assessing to what extent it meets those when implemented in the problem context. The central problem of treatment validation is that no real-world implementation is available to investigate whether the treatment contributes to stakeholder goals. Still, we want to predict what will happen if the treatment is implemented [46]. An expert opinion was used complemented by results obtained during the exploratory research. First the setup and validation goal is discussed in section 8.1. In section 8.2 the expert opinion is presented.

8.1 Validation setup

Using expert opinions and based on the exploratory research the proposed model for RL-based business process re-engineering is validated. Because a large portion of the activities in the proposed model are also performed during the exploratory research, the results are also part of the model validation. Almost all activities in the first two phases are performed during the exploratory research, the validated activities can be found in Figure 8.2. The activities not covered by the exploratory research are validated by the team lead from the Logistics Support department of Albert Heijn. The validation approach per phase is visualized in Figure 8.1.

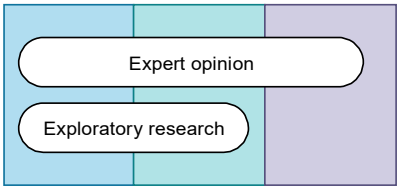


Figure 8.1: Validation for each phase in the model

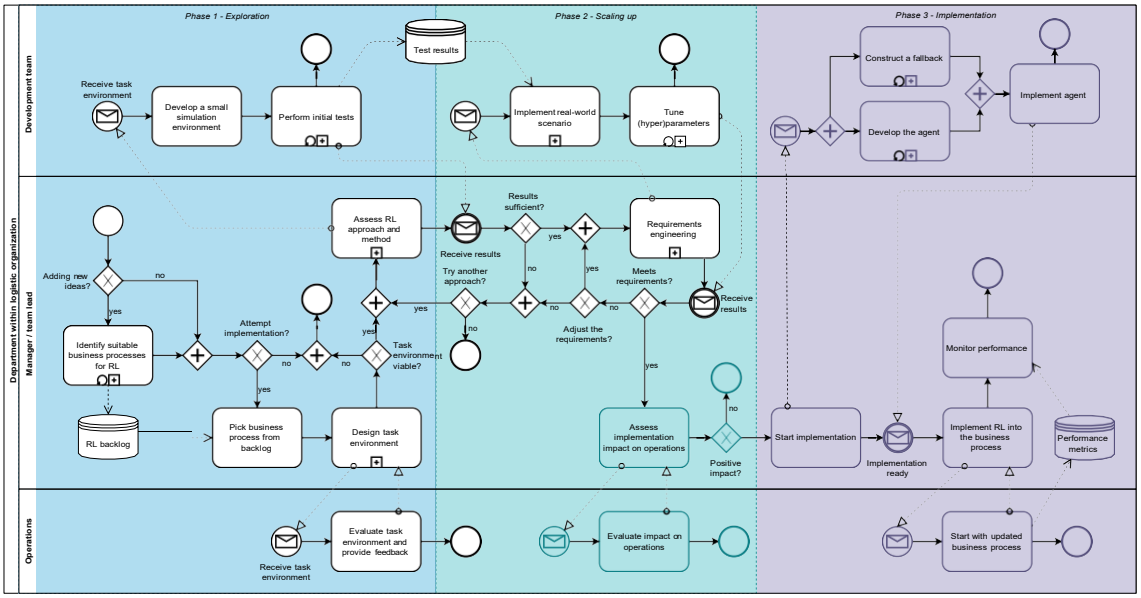


Figure 8.2: The activities also performed during the exploratory research

For each of the phases in the proposed model, the requirements defined in chapter 6 are validated by the team lead at the Logistics Support department at Albert Heijn. The goal of the validation is to develop a design theory of an artifact in context that allows us to predict what would happen if the artifact were transferred to its intended problem context [46].

8.2 Team lead expert opinion

The decision to use BPMN for the model makes the model easily adaptable by logistic organizations. The stakeholders are general to enable organizations to map their own teams and individuals on these roles.

The team lead points out that depending on the target organization, it is difficult to assess whether or not the team lead has the required knowledge in order to identify suitable business processes for RL. The team lead may lack in-depth knowledge about a process carried out by the operational staff each and every day that it is required to identify these processes. More examples in future RL implementations could ease the identification. A team lead might also decide to get the operational staff involved beforehand.

During the exploratory research the Logistics Support team got an introduction into artificial intelligence and reinforcement learning. When an AI novice organization would attempt to re-engineer their business processes with RL having only the model would not be enough. The examples from the exploratory research aid this problem to some extent, however these organizations will need a proper introduction into the field before commencing. The introduction helps to get the conversation of AI within the department started but it is important to give examples the employees can relate to. Examples such as AGVs in the warehouses are well known but RL can also solve less visible tasks such as the product allocation.

The model consists of multiple moments where the team lead can make the decision whether or not to continue with reinforcement learning. Being able to start small and slowly scale up is really valuable according to the team lead, as it does not require huge up-front investments both in terms of costs and scarce IT personnel.

The team lead suggests that when the business process is identified it makes sense to concurrently get the developers involved to design the task environment together. This alleviates the problem of the developers being unable to develop the environment thought out by the team lead.

The expert notes that having an idea about the workload of every stakeholder during the implementation process helps in the decision making about whether or not to continue. Maybe more important, it gives an idea about whether the team lead is allocating enough resources during each phase.

Based on the expert opinion and the exploratory research logistic organizations using the proposed model are able to re-engineer their business processes using RL, but there are some preconditions to be met. AI Novice organizations do need a proper introduction into the field of AI and the characteristics of RL before proceeding. Without the examples the model could be difficult to interpret, especially when tackling a very different business process compared to the product allocation problem.

V

Closure

9. Conclusion

In this chapter the research is concluded and an overview of the contribution of this thesis for both practice and literature is presented. Finally the limitations and future work are discussed.

What is the current state of artificial intelligence and especially deep and reinforcement learning in the logistics industry?

Artificial Intelligence (AI) will become very important for businesses across the world, and its time is now. Organizations that successfully implement AI are said to profit disproportionately compared to the laggards. A literature review on both AI and technology adoption shows that logistic organizations struggle to implement AI because of its unique determinants as well as not having clear and concise tools to improve their AI maturity. Deep learning is currently the most promising AI technique but reinforced learning is also gaining momentum. Reinforcement learning encompasses all of AI, an agent learns by performing actions in an environment and eventually finds an optimal policy to maximize its reward. This resembles in a lot of ways humans learn, although currently not being very efficient. Reinforcement learning became much more powerful due to the addition of neural networks and multiple approaches and algorithms have been identified, such as Deep Q-learning (DQN) and Advantage Actor Critic (A2C). Reinforcement learning literature has skyrocketed in recent years due to breakthroughs such as defeating the world champion in a game of Go. But little is known about whether and how this technique could be implemented into the business processes of logistic organizations.

What are the most important artificial intelligence adoption models and frameworks in the logistics industry?

Technology adoption research has been around for a long time, and through time multiple acceptance models were developed. Starting with the TAM which was eventually extended and resulted in the UTAUT model. The UTAUT model is often criticized because its determinants are not entirely compatible with AI. This is especially true because of the potential job loss and the disruptive nature of AI. In practice therefore, most of the time the TOE model is used to draw conclusions regarding the adoption of AI in organizations. Other adoption models such as the DOI are still relevant for AI as it follows a similar pattern and the potential rewards for early adopters is disproportionate. The TOE model was extended by Mahroof with perceived benefits as this was a great predictor for AI adoption in a large retailer warehouse. Based on the task at hand, an assessment can be made whether or not AI is suitable following the intelligence amplification framework. The framework shows what type of task humans excel at and what tasks are better off handled by computers.

Which types of business processes are suitable for reinforcement learning?

The literature body helped in identifying most of the characteristics a business process should have to be suitable for RL. During the exploratory research three potential business processes were considered and based on the results of the literature review the product allocation in the warehouses of Albert Heijn was picked because of its deterministic, fully observable nature. After implementing real-world scenarios of the slotting and the difficulties along the way the list with characteristics was refined and incorporated in the guidelines of the final model. Logistic organizations that want to identify a suitable business process to re-engineer with RL should also consider the size of the task environment and if rewards are sparse or immediate. The agent trained in the exploratory research has to be retrained when locations or product specifications change, picking a business process that does not change in terms of environment and actions the agent can take are also more suitable.

Which steps help logistic organizations in successfully implementing reinforcement learning?

During the exploratory research an attempt was made to develop a RL agent which is able to allocate products successfully in a small but realistic warehouse setting. The slotting of products is an important and labour intensive task and currently performed by multiple full-time employees.

Traditional machine learning approaches are only partially applicable to RL such as (hyper)parameter optimization and deciding whether or not there is a business case. In RL however, these tasks are only a subset of a set of tasks required to even get started. First a task environment needs to be identified and created for the agent to interact with. This can seem straightforward with a deterministic environment such as a chess board but when considering business processes in logistic organizations this could be very difficult. One hot encoding is one technique to pass an environment such as a spreadsheet to an agent used successfully in this thesis. Another important and difficult task in RL is the designing of the reward function, this often requires in-depth analysis together with the operational staff. For

the slotting a scoreboard was developed that was created in cooperation with the employees in an attempt to give the agent proper rewards. The reward function should be detailed enough for the agent to learn the correct policy however when over-engineering the agent could not learn at all. As with other high risk projects it is important to start small with a manageable task environment and scale up from there as the agent continues to learn.

To what extent can the developed model help logistic organizations in the adoption of reinforcement learning?

Based on the exploratory research and the literature body a BPMN model was developed together with a set of guidelines that enables logistic organizations to re-engineer their business processes using reinforcement learning. Although being aimed at AI novice and AI ready organizations, there are some preconditions for organizations before commencing. There should be a basic understanding about AI and RL across the entire department in order to get started with the implementation. This is because the team lead - the stakeholder responsible for the implementation - needs both the developers and the operational staff in its decision process. RL differs from supervised and unsupervised learning as it is able to solve a wide variety of tasks with the same agent, however it succeeds in only a fraction compares to the mainstream learning methods. The model therefore includes multiple moments in which the team lead can decide whether or not to continue with the implementation.

The business process targeted for the validation was the slotting process at Albert Heijn. Based on a realistic case, numerous staff members participated in an experiment to find out whether or not the product allocation performance could be increased using an RL agent. The experiment showed that using intelligence amplification, in which the staff members worked in cooperation with the RL agent significantly reduced the required effort while still achieved good performance. The results showed that although the staff was able to reach the maximum score, the agent was not far off. When the agent was asked to perform the entire task by itself, the agent was able to find a local optimum in almost every attempt.

The scientific relevance is twofold. Current adoption models lack the unique determinants for artificial intelligence and reinforcement learning, the methodology of this research could alleviate this problem for future research. Secondly, this research also indicates that using intelligence amplification, agents using reinforcement learning also benefit from the cooperation between a human and the agent. The model can be considered a first step in taking reinforcement learning beyond simple games and towards actual business processes.

9.1 Limitations and future work

Even though literature of reinforcement learning has only recently gained a lot of attention, the field exists for a long time. Because of the limited period in which the literature review was written, not all relevant contributions are considered. Since the development of Advantage Actor Critic (A2C) AI organizations have developed more state-of-the-art algorithms that are beyond the scope of this research.

The main focus of this research was on developing an agent capable of being implemented in a business process. Due to time constraints the proposed model the treatment was only validated using expert opinions and the exploratory research. The model without the exploratory research examples is currently not sufficient for AI novice organizations, future work can extend the number of implementations and further develop the model.

When developing the agent, due to computational limitations the largest scenario considered consists of 42 locations and 29 products to allocate, whereas in the real-world the slotters work with environments that could consist thousands of locations. Future work could use the model and attempt an implementation at a logistic organization that encompasses all three phases.

9.2 Recommendations for Albert Heijn

Because the agent used for the product allocation is model-free, it can be used for a wide variety of problems. The team lead and the developers can continue to experiment with the A2C algorithm and when new state-of-the-art RL algorithms are developed the department can quickly adapt because the task environment and reward function do not have to change.

Even though reinforcement learning has not been able to solve the slotting perfectly or outperform the traditional way of working does not mean it could not add value. Reevaluating tasks within the supply chain of Albert Heijn could identify tasks where agents would excel at and tasks better suited for humans. The application of the intelligence amplification framework improves this identification process.

Postface

References

[1] Accenture, SAS, and Intel. "AI Momentum, Maturity & Models for success". In: (2018), p. 24 (cit. on pp. 60–63).

[2] Icek Ajzen and Martin Fishbein. *The Prediction of Behavior from Attitudinal and Normative Variables*. Tech. rep. 1970, pp. 466–487 (cit. on pp. 55, 128).

[3] Abdulrahman Altahhan. "Self-reflective deep reinforcement learning". In: *Proceedings of the International Joint Conference on Neural Networks* 2016-Octob.8 (2016), pp. 4565–4570. ISSN: 1053-5888. DOI: 10.1109/IJCNN.2016.7727798 (cit. on pp. 46, 125).

[4] Alex M. Andrew. "Reinforcement Learning: An Introduction". In: *Kybernetes* 27.9 (1998), pp. 1093–1096. ISSN: 0368492X. DOI: 10.1108/k.1998.27.9.1093.3 (cit. on pp. 47–49, 51, 52, 125).

[5] Richard Bellman. *A Markovian Decision Process*. 1957. DOI: 10.1512/iumj.1957.6.56038 (cit. on pp. 46, 125).

[6] Ben Gesing, Steve J. Peterson, and Dirk Michelsen. "Artificial Intelligence in Logistics - A collaborative report by DHL and IBM on implications and use cases for the logistics industry". In: *DHL Customer Solutions & Innovation* (2018), p. 45. URL: https://www.logistics.dhl/content/dam/dhl/global/core/documents/pdf/glo- ai- in-logistics-white-paper.pdf (cit. on pp. 23, 24, 36, 60–62).

[7] Matthew Botvinick et al. "Reinforcement Learning, Fast and Slow". In: *Trends in Cognitive Sciences* 23.5 (2019), pp. 408–422. ISSN: 1879307X. DOI: 10.1016/j.tics.2019.02.006. URL: https://doi.org/10.1016/j.tics.2019.02.006 (cit. on p. 125).

[8] E. S. Brunette, R. C. Flemmer, and C. L. Flemmer. "A review of artificial intelligence". In: *ICARA 2009 - Proceedings of the 4th International Conference on Autonomous Robots and Agents*. 2009, pp. 385–392. ISBN: 9781424427130. DOI: 10.1109/ICARA.2000.4804025 (cit. on pp. 29, 36).

[9] Jacques Bughin et al. "Notes from the AI frontier: Modeling the global economic impact of AI". In: *McKinsey Global Institute* September.September (2018), pp. 1–64. URL: https://www.mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-ai-frontier-modeling-the-impact-of-ai-on-the-world-economy (cit. on pp. 23, 36).

[10] Lucian Bus¸oniu et al. "Reinforcement learning for control: Performance, stability, and deep approximators". In: *Annual Reviews in Control* 46 (2018), pp. 8–28. ISSN: 13675788. DOI: 10.1016/j.arcontrol.2018.09.005 (cit. on pp. 46, 53, 125).

[11] Fariborz Damanpour and Marguerite Schneider. "Phases of the adoption of innovation in organizations: Effects of environment, organization and top managers". In: *British Journal of Management* 17.3 (Sept. 2006), pp. 215–236. ISSN: 10453172. DOI: 10.1111/j.1467-8551.2006.00498.x (cit. on pp. 57, 128).

[12] Fred D. Davis. "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology". In: *MIS Quarterly* 13.3 (Sept. 1989), p. 319. ISSN: 02767783. DOI: 10.2307/249008. URL: https://www.jstor.org/stable/249008?origin=crossref (cit. on pp. 55, 56, 128).

[13] Fred D. Davis, Richard P.Bagozzi, and Paul R. Warshaw. "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models". In: *Management Science* 35.8 (Aug. 1989), pp. 982–1003. ISSN: 0025-1909. DOI: 10.1287/mnsc.35.8.982. URL: http://pubsonline.informs.org/doi/abs/10.1287/mnsc.35.8.982 (cit. on pp. 56, 128).

[14] Peter Dayan and Yael Niv. "Reinforcement learning: The Good, The Bad and The Ugly". In: *Current Opinion in Neurobiology* 18.2 (2008), pp. 185–196. ISSN: 09594388. DOI: 10.1016/j.conb.2008.08.003 (cit. on pp. 46, 49, 53, 125).

[15] Li Deng. "Artificial Intelligence in the Rising Wave of Deep Learning: The Historical Path and Future Outlook". In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 177–180. ISSN: 10535888. DOI: 10.1109/MSP.2017.2762725 (cit. on pp. 23, 36, 45, 46, 125).

[16] Rocco DePietro, Edith Wiarda, and Mitchell Fleischer. "The context for change: Organization, Technology and Environment". In: *The process of technology innovation*. 1990 (cit. on pp. 58, 59, 94, 128).

[17] Andrej Dobrkovic et al. "Intelligence amplification framework for enhancing scheduling processes". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10022 LNAI. Springer Verlag, 2016, pp. 89–100. ISBN: 9783319479545. DOI: 10.1007/978-3-319-47955-2{_}8 (cit. on pp. 59, 85, 128).

[18] Marta Garnelo and Murray Shanahan. "Reconciling deep learning with symbolic artificial intelligence: representing objects and relations". In: *Current Opinion in Behavioral Sciences* 29 (2019), pp. 17–23. ISSN: 23521546. DOI: 10.1016/j.cobeha.2018.12.010. URL: https://doi.org/10.1016/j.cobeha.2018.12.010 (cit. on pp. 46, 125).

[19] Karen Hao. *We analyzed 16,625 papers to figure out where AI is headed next*. 2019. URL: https://www.technologyreview.com/s/612768/we-analyzed-16625-papers-to-figure-out-where-ai-is-headed-next/ (cit. on p. 23).

[20] Nicholas R Jennings and Michael Wooldridge. *A Roadmap of Agent Research and Development*. Tech. rep. 1998, pp. 7–38 (cit. on pp. 23, 36).

[21] Elena Karahanna, Detmar W Straub, and Norman L Chervany. *Information Technology Adoption Across Time: A Cross-Sectional Comparison of Pre-Adoption and Post-Adoption Beliefs*. Tech. rep. 2. 1999, pp. 183–213. URL: http://www.jstor.orgStableURL:http://www.jstor.org/stable/249751 (cit. on pp. 56, 128).

[22] Barbara Kitchenham. *Procedures for Performing Systematic Reviews*. Tech. rep. 2004 (cit. on p. 29).

[23] Matthias Klumpp and Henk Zijm. "Logistics Innovation and Social Sustainability: How to Prevent an Artificial Divide in Human–Computer Interaction". In: *Journal of Business Logistics*. Wiley-Blackwell, 2019. DOI: 10.1111/jbl.12198 (cit. on pp. 60, 128).

[24] K. H. Leung et al. "Design of a case-based multi-agent wave picking decision support system for handling e-commerce shipments". In: *PICMET 2016 - Portland International Conference on Management of Engineering and Technology: Technology Management For Social Innovation, Proceedings*. 2017. ISBN: 9781509035953. DOI: 10.1109/PICMET.2016.7806645 (cit. on pp. 60, 128).

[25] Long Ji Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Machine Learning* 8.3 (1992), pp. 293–321. ISSN: 15730565. DOI: 10.1023/A:1022628806385 (cit. on pp. 51, 125).

[26] Kamran Mahroof. "A human-centric perspective exploring the readiness towards smart warehousing: The case of a large retail distribution warehouse". In: *International Journal of Information Management* 45 (Apr. 2019), pp. 176–190. ISSN: 02684012. DOI: 10.1016/j.ijinfomgt.2018.11.008 (cit. on pp. 23, 55, 60, 128).

[27] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (2013), pp. 1–9. URL: http://arxiv.org/abs/1312.5602 (cit. on pp. 50, 125).

[28] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *33rd International Conference on Machine Learning, ICML 2016* 4 (2016), pp. 2850–2869 (cit. on pp. 52, 53, 125).

[29] Gary C Moore and Izak Benbasat. *Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation*. Tech. rep. 1991 (cit. on pp. 56, 128).

[30] Tomaso Poggio et al. "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review". In: *International Journal of Automation and Computing* 14.5 (2017), pp. 503–519. ISSN: 17518520. DOI: 10.1007/s11633-017-1054-2 (cit. on pp. 45, 125).

[31] Tom Pringle and Eden Zoller. "How to Achieve AI Maturity and Why It Matters". In: (2018), p. 18. URL: https://www.amdocs.com/sites/default/files/filefield_paths/ai-maturity-model-whitepaper.pdf (cit. on pp. 63, 67, 94).

[32] Yara Rizk et al. "Deep belief networks and cortical algorithms: A comparative study for supervised classification". In: *Applied Computing and Informatics* 15.2 (2018), pp. 81–93. ISSN: 22108327. DOI: 10.1016/j.aci.2018.01.004. URL: https://doi.org/10.1016/j.aci.2018.01.004 (cit. on pp. 45, 125).

[33] Everett M. Rogers. *Diffusion of innovations*. Free Press, 1983, p. 453. ISBN: 0029266505 (cit. on pp. 56, 57, 59, 102, 128).

[34] Maria Rosario Oliveira Martins, Tiago Oliveira, and Maria Fraga Martins. "Literature Review of Information Technology Adoption Models at Firm Level". In: *The Electronic Journal Information Systems Evaluation* 14 (2011), p. 110. ISSN: 1566-6379 (cit. on pp. 55, 128).

[35] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach Third Edition*. Tech. rep. 2010, p. 1151. DOI: 10.1017/S0269888900007724. arXiv: 9809069v1 [gr-qc] (cit. on pp. 35–49, 123).

[36] Tom Schaul et al. "Prioritized Experience Replay". In: (2015), pp. 1–21. URL: http://arxiv.org/abs/1511.05952 (cit. on pp. 51, 125).

[37] Jürgen Schmidhuber. "Deep Learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 18792782. DOI: 10.1016/j.neunet.2014.09.003. URL: http://dx.doi.org/10.1016/j.neunet.2014.09.003 (cit. on pp. 43, 45, 46, 125).

[38] V. V. Shakirov, K. P.Solovyeva, and W. L. Dunin-Barkowski. "Review of State-of-the-Art in Deep Learning Artificial Intelligence". In: *Optical Memory and Neural Networks* 27.2 (2018), pp. 65–80. ISSN: 1060-992X. DOI: 10.3103/s1060992x18020066 (cit. on pp. 46, 125).

[39] Ajay Shrestha and Ausif Mahmood. "Review of Deep Learning Algorithms and Architectures". In: *IEEE Access* 7 (2019), pp. 53040–53065. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2912200. URL: https://ieeexplore.ieee.org/document/8694781/ (cit. on pp. 45, 125).

[40] Vivienne Sze et al. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329. ISSN: 00189219. DOI: 10.1109/JPROC.2017.2761740. URL: http://ieeexplore.ieee.org/document/8114708/ (cit. on pp. 45, 125).

[41] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double Q-Learning". In: *30th AAAI Conference on Artificial Intelligence, AAAI 2016*. 2016. ISBN: 9781577357605 (cit. on pp. 51, 125).

[42] Venkatesh et al. "User Acceptance of Information Technology: Toward a Unified View". In: *MIS Quarterly* 27.3 (2003), p. 425. ISSN: 02767783. DOI: 10.2307/30036540. URL: https://www.jstor.org/stable/10.2307/30036540 (cit. on pp. 58, 128).

[43] Yuan Wang et al. "Enhancing transportation systems via deep learning: A survey". In: *Transportation Research Part C: Emerging Technologies* 99.December 2018 (2019), pp. 144–163. ISSN: 0968090X. DOI: 10.1016/j.trc.2018.12.004. URL: https://doi.org/10.1016/j.trc.2018.12.004 (cit. on pp. 45, 125).

[44] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3-4 (1992), pp. 279–292. ISSN: 0885-6125. DOI: 10.1007/bf00992698 (cit. on pp. 49, 50, 75, 125).

[45] Jane Webster and Richard T Watson. *Analyzing the Past to Prepare for the Future: Writing a Literature Review*. Tech. rep. 2. 2002, pp. xiii–xxiii (cit. on pp. 28, 30, 124, 126, 127).

[46] Roel J Wieringa and Software Engineering. *Design Science Methodology*. ISBN: 9783662438381 (cit. on pp. 26–28, 31, 93, 94, 107, 108).

[47] Joost F. Wolfswinkel, Elfi Furtmueller, and Celeste P.M. Wilderom. *Using grounded theory as a method for rigorously reviewing literature*. Jan. 2013. DOI: 10.1057/ejis.2011.51 (cit. on pp. 29, 30, 124, 127).

[48] Yue-ting Zhuang et al. "Challenges and opportunities: from big data to knowledge in AI 2.0". In: *Frontiers of Information Technology & Electronic Engineering* 18.1 (2017), pp. 3–14. ISSN: 2095-9184. DOI: 10.1631/fitee.1601883 (cit. on pp. 36, 125).

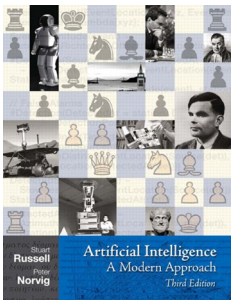
Appendices

A Literature review results of reinforcement learning

For this topic one of the most influential books on AI is used first and appended by a structured literature review on recent review papers regarding - state of the art - deep learning and reinforcement learning. The aim of this review is therefore to get an overview of the current state of artificial intelligence and reinforcement learning literature. First the book is discussed followed by the details and results of the structured literature review.

Book

The foundation for this research topic is the highly cited book "Artificial Intelligence - A Modern Approach" by Russel and Norvig because it forms a proper foundation of the field [35].



The 1152 pages offer one of the most comprehensive, up-to-date introduction to the theory and practice of AI and is therefore a great starting point for diving into AI.

SLR

In addition a literature review was conducted to find relevant and more recent contributions in the field. Because the book of Russel and Norvig covers AI in general, the focus of the SLR was on deep learning that currently delivers the most promising results and a look into the future with reinforced learning.

After an initial search and the relevance of the results the following search queries

were used to find relevant literature:

- TITLE-ABS-KEY ("neural network*")
  n: 423.706
- TITLE-ABS-KEY ("deep learning")
  n: 39.666
- TITLE-ABS-KEY ("reinforce* learning")
  n: 198

Because of the huge number of articles on the topic, inclusion and exclusion criteria have been defined in an attempt to improve and refine the literature body.

Inclusion criteria:

- English peer reviewed studies.
- Review papers that are related to deep learning or reinforced learning.

Exclusion criteria:

- Studies that are not accessible.
- Studies that are not related to the research questions.
- Duplicate studies.
- Short papers.
- Studies in which deep learning or reinforcement learning is not the main topic.

Results

This literature study was conducted on the 18th of June 2019. The results can be found in Table A.1.

n_1	n_2	n_3	n_4	n_5
441.485	319	50	21	22

Table A.1: Number of results for the reinforcement learning SLR

Using the extended concept matrix the articles and their concepts have been identified [45, 47].

The final selection of articles can be found in Table A.2. How each concept relates to the research questions is visualized in Table A.3.

Papers	**Concepts**				
	History and future of DL	From shallow towards deep learning	Challenges and opportunities of DL	DL applications and techniques	Deep reinforced learning
Poggio et al. (2017) [30]		**x**	**x**		
Wang et al. (2019) [43]				**x**	
Rizk et al. (2018) [32]	**x**	**x**		**x**	
Garnelo et al. (2019) [18]			**x**	**x**	
Bus¸oniu et al. (2018) [10]			**x**		**x**
Botvinick et al. (2019) [7]	**x**		**x**	**x**	**x**
Schmidhuber (2015) [37]	**x**	**x**	**x**	**x**	**x**
Sze et al. (2017) [40]			**x**		
Zhuang et al. (2017) [48]	**x**		**x**		
Shrestha et al. (2019) [39]				**x**	
Altahhan et al. (2016) [3]				**x**	**x**
Shakirov et al. (2018) [38]	**x**		**x**		
Deng (2018) [15]	**x**		**x**		
Mnih et al. (2013) [27]					**x**
Mnih et al. (2016) [28]					**x**
Andrew et al. (1998) [4]					**x**
Van Hasselt et al. (2016) [41]					**x**
Schaul et al. (2015) [36]					**x**
Lin et al. (1992) [25]					**x**
Dayan et al. (2008) [14]					**x**
Bellman et al. (1957) [5]					**x**
Watkins et al. (1992) [44]					**x**

Table A.2: The concept matrix for deep learning and reinforced learning

Concept	SQ1	SQ2
History and future of DL	x	x
From shallow towards deep learning	x	
Challenges and opportunities of DL	x	x
DL applications and techniques	x	
Deep reinforced learning	x	

Table A.3: AI concepts

B Literature review results of technology adoption

Technology acceptance and acceptance on both individual level as well as on an organizational level has been a subject of research since the late 1980s. Because of the abundance and the persisting relevance of articles a twofold methodology was used. First technology adoption is considered and papers were selected based on recommendations of experts of the University of Twente. To be able to compare technologies or innovations to artificial intelligence a structured literature review was used to find articles on AI adoption in preferably logistic organizations.

Experts

Contacting senior experts on a particular topic in conducting a literature review is of great importance in order to get a high-quality review [45]. For this review experts and lecturers at the University of Twente were contacted to identify the most influential articles, the foundation of the field. The experts that contributed articles to this literature review are professor M.E. Iacob and associate professor M.J. van Sinderen. The selection consists mainly of technology acceptance models and how they evolved over time. Research that was added to the review using this method are marked with an asterisk in the results, which can be found in Table B.1.

SLR

Whereas the expert articles form the foundation of the topic, more recent literature on technology - and particularly AI - adoption were needed to be able to answer the research questions. Therefore the expert articles are complemented with a SLR.

Based on the relevance of the most cited results for different search queries, the following search query was used:

. TITLE-ABS-KEY ("artificial intelligence" OR "AI" OR "intelligent agent*") AND ("adopt*" OR "accept*") AND "logistic*"
n: 307

The following inclusion and exclusion criteria were used for technology adoption:

Inclusion criteria:

- English peer reviewed studies.
- Published between 2000 and 2019.
- Related to technology - or AI - adoption (in logistical organizations).

- Acceptance articles on both organizational and individual level.

Exclusion criteria:

- Articles that are not accessible.
- Articles that are not peer reviewed.
- Duplicate studies.
- Short papers.
- Studies that have a very narrow scope.

Results

This literature study was conducted on the 15th of May 2019. The results can be found in Table B.1.

n_1	n_2	n_3	n_4	n_5
302	712	33	14	14

Table B.1: Number of results for technology adoption SLR

The following concepts emerged from the literature body:

- Technology adoption at an individual level.
- Technology adoption at organizational level.
- Artificial intelligence adoption.
- Innovation diffusion.
- Implementation of AI.
- Human-computer symbiosis.

Using the extended concept matrix the articles and their concepts have been identified [45, 47]. The final selection of articles can be found in Table B.2. How each concept relates to the research questions is visualized in Table B.3.

Papers	**Concepts**					
	Technology adoption at individual level	Technology adoption at organizational level	Artificial Intelligence adoption	Innovation diffusion	Implementation of AI	Human-computer symbiosis
Karahanna et al. (1999) [21]	**x**					
Moore et al. (1991) [29]	**x**			**x**		
Venkatesch et al. (2003)* [42]						
Rogers (1983)* [33]	**x**	**x**		**x**		
DePietro et al. (1990) [16]	**x**	**x**		**x**		
Ajzen et al. (1970)* [2]	**x**					
Davis (1989)* [12]	**x**					
Davis et al. (1989)* [13]	**x**					
Dobrkovic (2016)* [17]	**x**		**x**			**x**
Damanpour et al. (2006) [11]		**x**		**x**		
Mahroof et al. (2019) [26]	**x**	**x**	**x**		**x**	
Klumpp et al. (2019) [23]	**x**		**x**			**x**
Oliveira et al. (2011) [34]	**x**	**x**		**x**		
Leung et al. (2016) [24]					**x**	

Table B.2: The concept matrix for technology adoption

Concept	**SQ1**	**SQ2**
Technology adoption at individual level		**x**
Technology adoption at organizational level		**x**
Artificial intelligence adoption	**x**	**x**
Innovation diffusion		**x**
Implementation of AI	**x**	**x**
Human-computer symbiosis	**x**	**x**

Table B.3: Technology adoption concepts

C Scenarios

In this appendix the scenarios used in the exploratory research and intelligence amplification chapter are discussed.

Scenario A

Scenario A contains 10 locations and 6 products to slot. The products can be found in Table C.1, the initial slotting and the optimal slotting can be found in Table C.2. When slotted optimally, the resulting scoreboard can be found in Table C.3. The circuit can be found in Figure C.1.

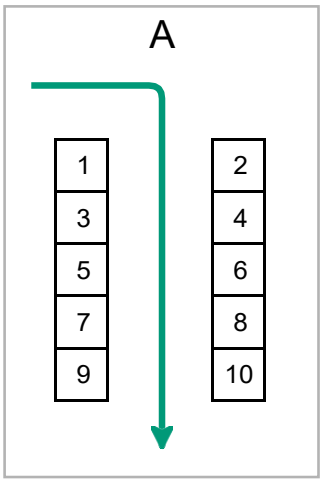


Figure C.1: The circuit for scenario A

Product	Units per pallet	Units in flow rack	SDF (+1)	SDF (+2)	SDF (+3)	Stacking group	Stacking class
Cookies	220	20	1500	2200	2200	D	14000
Soup	180	12	700	500	600	C	35000
Beer	40	-	800	800	1000	A	10000
Toilet paper	24	-	250	200	300	C	40000
Rice	120	20	700	700	550	D	60000
Coffee	150	15	1000	800	900	B	12000

Table C.1: Products to slot in scenario A

Location	Flow rack	Next location	Slotted initially	Optimal slotting
1	False	3	-	Beer
2	False	4	-	Coffee
3	False	5	-	Beer
4	False	6	-	Cookies
5	False	7	-	Beer
6	False	8	-	Cookies
7	False	9	-	Soup
8	False	10	-	Toilet paper
9	False	-	-	Rice
10	False	-	-	Toilet paper

Table C.2: Locations and optimal slotting for scenario A

Reward	Score	Occurrences	Total
Product slotted (A)	+15	1	15
Product slotted (B)	+15	1	15
Product slotted (C)	+15	2	30
Product slotted (D)	+15	2	30
Free locations	+2	0	0
Matching SDF (+1)	+15	6	90
Matching SDF (+2)	+10	6	60
Matching SDF (+3)	+5	5	25
Movement	-1	10	-10
Facings not adjacent	-15	0	0
Stacking group violation	-15	0	0
Stacking class violation	-10	0	0
			255

Table C.3: The scoreboard for the optimal slotting in scenario A

Scenario B

Scenario B contains 20 locations and 13 products to slot. The products can be found in Table C.4, the initial slotting and the optimal slotting can be found in Table C.5. When slotted optimally, the resulting scoreboard can be found in Table C.6. The circuit can be found in Figure C.2.

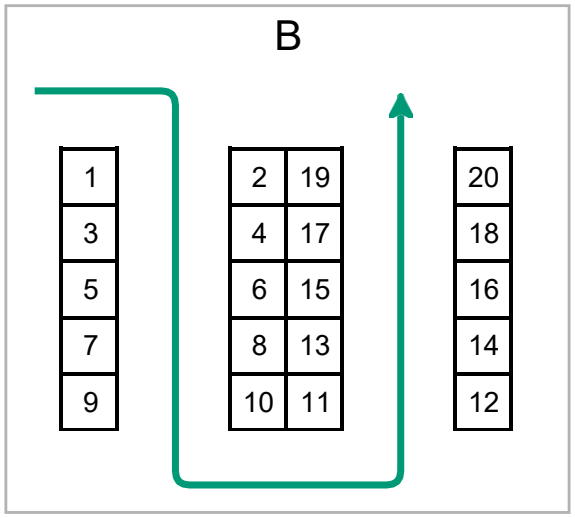


Figure C.2: The circuit for scenario B

Product	Units per pallet	Units in flow rack	SDF (+1)	SDF (+2)	SDF (+3)	Stacking group	Stacking class
Ketchup	120	10	700	700	550	D	53000
Cereal	60	-	200	500	250	D	52000
Rice	120	20	700	700	550	C	40000
Pasta	25	5	100	120	90	D	57000
Coffee	150	15	1000	800	900	B	12000
Eggs	90	20	300	200	550	C	42000
Apple juice	70	-	500	450	500	D	12000
Toilet paper	24	-	120	140	160	D	60000
Candy	120	40	700	700	550	C	42000
Soda	40	-	400	800	700	B	11000
Soup	180	12	1000	2500	2500	C	25000
Beer	40	-	800	800	1000	A	10000
Cookies	220	20	1200	1500	1200	D	14000

Table C.4: Products to slot in scenario B

Location	Flow rack	Next location	Slotted initially	Optimal slotting
1	False	3	-	Beer
2	False	4	-	Soda
3	False	5	-	Beer
4	False	6	-	Soda
5	False	7	-	Beer
6	False	8	-	Soda
7	False	9	-	Beer
8	False	10	-	Coffee
9	False	-	-	Apple juice
10	False	-	-	Cookies
11	False	13	-	Soup
12	False	14	-	Rice
13	False	15	-	Soup
14	False	16	-	Candy
15	False	17	-	Eggs
16	False	18	-	Cereal
17	False	19	-	Ketchup
18	False	20	-	Cereal
19	False	-	-	Pasta
20	False	-	-	Toilet paper

Table C.5: Locations and optimal slotting for scenario B

Reward	Score	Occurrences	Total
Product slotted (A)	+15	1	15
Product slotted (B)	+15	2	30
Product slotted (C)	+15	4	60
Product slotted (D)	+15	6	90
Free locations	+2	0	0
Matching SDF (+1)	+15	12	180
Matching SDF (+2)	+10	13	130
Matching SDF (+3)	+5	12	60
Movement	-1	20	-20
Facings not adjacent	-15	0	0
Stacking group violation	-15	0	0
Stacking class violation	-10	0	0
			545

Table C.6: The scoreboard for the optimal slotting in scenario B

Scenario C

Scenario C contains 42 locations and 29 products to slot. The products can be found in Table C.8, the initial slotting and the optimal slotting can be found in Table C.9. When slotted optimally, the resulting scoreboard can be found in Table C.7. The circuit can be found in Figure C.3.

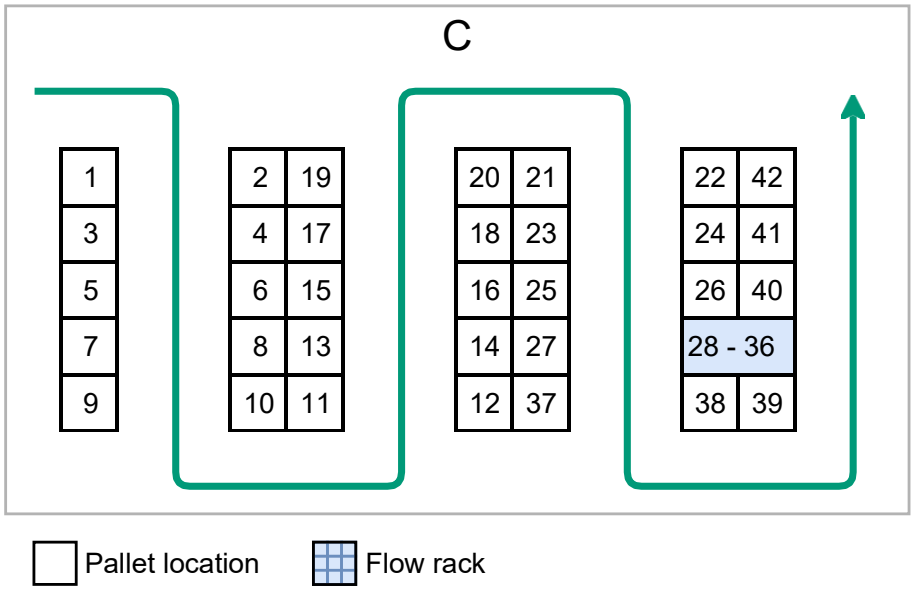


Figure C.3: The circuit for scenario C

Reward	Score	Occurrences	Total
Product slotted (A)	+15	1	15
Product slotted (B)	+15	2	30
Product slotted (C)	+15	4	60
Product slotted (D)	+15	22	330
Free locations	+2	0	0
Matching SDF (+1)	+15	28	420
Matching SDF (+2)	+10	29	290
Matching SDF (+3)	+5	28	140
Movement	-1	42	-42
Facings not adjacent	-15	0	0
Stacking group violation	-15	0	0
Stacking class violation	-10	0	0
			1243

Table C.7: The scoreboard for the optimal slotting in scenario C

Product	Units per pallet	Units in flowrack	SDF (+1)	SDF (+2)	SDF (+3)	Stacking group	Stacking class
Beer	40	0	800	800	1000	A	10000
Soda	40	0	400	800	700	B	11000
Coffee	150	15	1000	800	900	B	12000
Apple juice	70	0	500	450	500	D	12000
Cookies	220	20	1200	1500	1200	D	14000
Soup	180	12	1000	2500	2500	C	25000
Rice	120	20	700	700	550	C	40000
Eggs	90	20	300	200	550	C	42000
Candy	120	40	700	700	550	C	42000
Cereal	60	0	200	500	250	D	52000
Ketchup	120	10	700	700	550	D	53000
Pasta	25	5	100	120	90	D	57000
Toilet paper	24	0	120	140	160	D	60000
Diapers	100	0	2000	2500	2500	D	60000
Sausages	80	20	400	400	550	D	62000
Bread	80	5	600	1100	1000	D	64000
Honey	400	40	60	50	50	D	64000
Chocolate	200	10	15	20	15	D	65000
Baby wipes	40	20	20	20	25	D	70000
Mustard	60	10	10	5	5	D	71000
Paprika powder	80	20	25	15	10	D	72000
Tissues	20	5	10	10	10	D	73000
Deodorant	50	20	30	40	40	D	80000
Peanut butter	40	10	10	5	20	D	90000
Almonds	60	10	10	15	10	D	90000
Apple pie	20	5	5	5	10	D	90000
Soap	50	5	10	10	5	D	90000
Sunscreen	200	40	800	700	800	D	92000
Crackers	40	5	400	800	600	D	95000

Table C.8: Products to slot in scenario C

Location	Flow rack	Next location	Slotted initially	Optimal slotting
1	False	3	-	Beer
2	False	4	-	Soda
3	False	5	-	Beer
4	False	6	-	Soda
5	False	7	-	Beer
6	False	8	-	Soda
7	False	9	-	Beer
8	False	10	-	Coffee
9	False	-	-	Apple juice
10	False	-	-	Cookies
11	False	13	-	Soup
12	False	14	-	Rice
13	False	15	-	Soup
14	False	16	-	Candy
15	False	17	-	Eggs
16	False	18	-	Cereal
17	False	19	-	Ketchup
18	False	20	-	Cereal
19	False	-	-	Pasta
20	False	-	-	Toilet paper
21	False	23	-	Diapers
22	False	24	-	Sausages
23	False	25	-	Diapers
24	False	26	-	Bread
25	False	27	-	Diapers
26	False	-	-	Bread
27	False	37	-	Diapers
28	True	-	-	Honey
29	True	-	-	Chocolate
30	True	-	-	Baby wipes
31	True	-	-	Mustard
32	True	-	-	Paprika powder
33	True	-	-	Tissues
34	True	-	-	Deodorant
35	True	-	-	Peanut butter
36	True	-	-	Almonds
37	False	-	-	Apple pie
38	False	-	-	Soap
39	False	-	-	Sunscreen
40	False	41	-	Crackers
41	False	42	-	Crackers
42	False	-	-	Crackers

Table C.9: Locations and optimal slotting for scenario C

D Advantage Actor-Critic agent Python implementation

The appendix is divided into three sections, first the installation requirements are discussed followed by the design of the agent. Finally the instructions to train and use the agent are shown.

Installation

To ensure that the Python code in this chapter can run properly on every machine a virtual environment is used. To install the virtual environment the following steps have to be followed:

1. Install pipenv (HTTPS://GITHUB.COM/PYPA/PIPENV).
2. Create a directory and copy the Pipfile found in Code segment 1[1].
3. Install the virtual environment, with the required packages by running the following command: **pipenv install**.
4. To run a python file using the virtual environment, one can run: "**pipenv shell filename.py**".

```
[[source]]
name = "pypi"
url = "https://pypi.org/simple"
verify_ssl = true

[dev-packages]

[packages]
numpy = "*"
pandas = "*"
gym = "*"
tensorflow = "==2.0.0-rc0"
progressbar2 = "*"
matplotlib = "*"
style = "*"
beautifultable = "*"

[requires]
python_version = "3.7"

[pipenv]
allow_prereleases = true
```

Code segment 1: The Pipfile used to create the virtual environment (Pipenv)

Design

The design of the A2C agent consists of the agent, its neural network and the environment.

Neural network

The implementation of the neural network (also called model) can be found in Code segment 2. Here the structure of the neural network is defined such as the number of actions the agent can take, the number of nodes (in a single layer) and the number of layers in the body. Once initialized these parameters are fixed to enable training an agent using the model multiple times. Other (hyper)parameters such as the learning rate are part of the agent as it decides how to update its internal model.

```
class ProbabilityDistribution(tf.keras.Model):
    def call(self,logits):
        return tf.squeeze(tf.random.categorical(logits,1),axis=-1)


```

[1]Depending on the task environment, one might not need all packages listed in the Pipfile.

```
class Model(tf.keras.Model):
    def __init__(self, locations, products, number_of_actions, number_of_nodes=512, number_of_extra_layers=0):
        super().__init__('mlp_policy')

        self.number_of_extra_layers = number_of_extra_layers

        self.flatten = kl.Flatten(input_shape=(locations, products + 1))
        self.hidden1 = kl.Dense(number_of_nodes, activation='relu')
        self.hidden2 = kl.Dense(number_of_nodes, activation='relu')

        # Extra layers
        if self.number_of_extra_layers == 1:
            self.hidden3 = kl.Dense(number_of_nodes, activation='relu')
        elif self.number_of_extra_layers == 2:
            self.hidden3 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden4 = kl.Dense(number_of_nodes, activation='relu')
        elif self.number_of_extra_layers == 3:
            self.hidden3 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden4 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden5 = kl.Dense(number_of_nodes, activation='relu')
        elif self.number_of_extra_layers == 4:
            self.hidden3 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden4 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden5 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden6 = kl.Dense(number_of_nodes, activation='relu')
        elif self.number_of_extra_layers == 5:
            self.hidden3 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden4 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden5 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden6 = kl.Dense(number_of_nodes, activation='relu')
            self.hidden7 = kl.Dense(number_of_nodes, activation='relu')

        self.value = kl.Dense(1, name='value')

        self.logits = kl.Dense(number_of_actions, name='policy_logits')
        self.dist = ProbabilityDistribution()

    def call(self, inputs):
        x = tf.convert_to_tensor(inputs, dtype=tf.float32)

        x = self.flatten(x)

        if self.number_of_extra_layers == 1:
            x = self.hidden3(x)
        elif self.number_of_extra_layers == 2:
            x = self.hidden3(x)
            x = self.hidden4(x)
        elif self.number_of_extra_layers == 3:
            x = self.hidden3(x)
            x = self.hidden4(x)
            x = self.hidden5(x)
        elif self.number_of_extra_layers == 4:
            x = self.hidden3(x)
            x = self.hidden4(x)
            x = self.hidden5(x)
            x = self.hidden6(x)
        elif self.number_of_extra_layers == 5:
            x = self.hidden3(x)
            x = self.hidden4(x)
            x = self.hidden5(x)
            x = self.hidden6(x)
            x = self.hidden7(x)

        hidden_logs = self.hidden1(x)
        hidden_vals = self.hidden2(x)

        return self.logits(hidden_logs), self.value(hidden_vals)

    def action_value(self, obs):
        logits, value = self.predict(obs)
        action = self.dist.predict(logits)
        return np.squeeze(action, axis=-1), np.squeeze(value, axis=-1)
```

Code segment 2: Neural network of A2C (model.py)

The "ProbabilityDistribution" class is used to randomly sample a categorical action. One could need to make some changes to the input as this differs with each task environment. The flatten layer is used to transform the multidimensional array of products on locations to a flattened array each value representing one input node. The *call* function is used to run input trough the model and return both the Q-values and the value of the current state. The *action_value* function is a helper method used later.

Agent

In order to test whether the agent learns two agents have been created, a baseline agent that makes random decisions and the actual agent. Using both agents and comparing the scores they are able to achieve one can assess whether or not the A2C agent performs better than when making random decisions. The baseline agent can be found in Code segment 3 and the A2C agent can be found in Code segment 3.

```
class RandomAgent:
    def __init__(self, model):
        self.model = model

    def test(self, env, render=True):
        obs, done, ep_reward = env.reset(), False, 0
        while not done:
            action, _ = self.model.action_value(obs[None, :])
            obs, reward, done, _ = env.step(action)
            ep_reward += reward
            if render:
                env.render()
        return ep_reward
```

Code segment 3: Baseline agent that picks actions randomly (random_agent.py)

Both agents are initialized using an instance of the Model class but the A2C agent can also take various (hyper)parameters. Such as the value function coefficient, entropy, gamma and the learning rate. The A2C agent includes a *train* method where the batch size and the number of updates can be passed as parameters. The *_returns_advantages* method that returns the advantages used during training.

```
import tensorflow as tf
import numpy as np
from progressbar import progressbar

import tensorflow.keras.losses as kls
import tensorflow.keras.optimizers as ko


class A2CAgent:
    def __init__(self, model, value=0.5, entropy=0.0001, gamma=0.95, learning_rate=0.0007):
        self.params = {'value': value, 'entropy': entropy, 'gamma': gamma}

        self.model = model
        self.model.compile(
            optimizer=ko.RMSprop(lr=learning_rate),
            loss=[self._logits_loss, self._value_loss]
        )

    def train(self, env, batch_sz=32, updates=500):
        rewards_list = np.array([])
        actions = np.empty((batch_sz,), dtype=np.int32)
        rewards, dones, values = np.empty((3, batch_sz))
        observations = np.empty((batch_sz,) + env.one_hot_encode().shape)
        ep_rews = [0.0]
        env.reset()
        next_obs = env.one_hot_encode()
        for update in progressbar(range(updates)):

            for step in range(batch_sz):
                observations[step] = env.one_hot_encode().copy()
                actions[step], values[step] = self.model.action_value(np.expand_dims(env.one_hot_encode(), axis=0))
                next_obs, rewards[step], dones[step], _ = env.step(actions[step])


                ep_rews[-1] += rewards[step]
                if dones[step]:
                    ep_rews.append(0.0)
                    rewards_list = np.append(rewards_list, env.get_score())
                    next_obs = env.reset()

            _, next_value = self.model.action_value(np.expand_dims(env.one_hot_encode(), axis=0))
            returns, advs = self._returns_advantages(rewards, dones, values, next_value)
            acts_and_advs = np.concatenate([actions[:, None], advs[:, None]], axis=-1)
            losses = self.model.train_on_batch(observations, [acts_and_advs, returns])

        return rewards_list

    def _returns_advantages(self, rewards, dones, values, next_value):
        returns = np.append(np.zeros_like(rewards), next_value, axis=-1)
        for t in reversed(range(rewards.shape[0])):
            returns[t] = rewards[t] + self.params['gamma'] * returns[t + 1] * (1 - dones[t])
        returns = returns[:-1]
        adv   antage = returns-   values
        return returns, advantages

```

```
    def test(self, env, render=True):
        env.reset()
        done, ep_reward = False, 0
        while not done:
            action, _ = self.model.action_value(np.expand_dims(env.one_hot_encode(), axis=0))
            obs, reward, done, _ = env.step(action)
            ep_reward += reward
            if render:
                env.render()
        print(env.get_score())
        return ep_reward

    def _value_loss(self, returns, value):
        return self.params['value'] * kls.mean_squared_error(returns, value)

    def _logits_loss(self, acts_and_advs, logits):
        actions, advantages = tf.split(acts_and_advs, 2, axis=-1)
        weighted_sparse_ce = kls.SparseCategoricalCrossentropy(from_logits=True)(actions
        = tf.cast(actions, tf.int32))
        policy_loss = weighted_sparse_ce(actions, logits, sample_weight=advantages)
        entropy_loss = kls.categorical_crossentropy(logits, logits, from_logits=True)
        return policy_loss - self.params['entropy'] * entropy_loss
```

Code segment 4: A2C agent implementation (agent.py)

Environment

The agent works with gym environments (HTTP://GYM.OPENAI.COM/DOCS/) created by OpenAI. It is possible however to create custom environments such as for the product allocation problem. Creating the (task) environment is completely dependent on the business process one wants to re-engineer using RL. This section focuses on the how to setup the environment and how to run the agent whereas section D highlights the implementation for the product allocation problem.

The scaffolding needed for custom gym environments can be found in Code segment 5.

```
gym-custom/
    README.md
    setup.py
    gym_custom/
        __init__.py
        envs/
            __init__.py
            custom_env.py
```

Code segment 5: The scaffolding for a custom gym environment

The setup file is presented in Code segment 6. Registering the environment is presented in Code 7, the identifier is used by the agent to initialize the environment.

```
from setuptools import setup

setup(name='gym_custom',
      version='0.0.1',
      install_requires=['gym']
)
```

Code segment 6: gym-custom/setup.py

```
from gym.envs.registration import register

register(
    id='custom-v0',
    entry_point='gym_custom.envs:CustomEnv',
)
```

Code segment 7: gym-custom/gym_custom/_init_.py

```
from gym_custom.envs.custom_env import CustomEnv
```

Code segment 8: gym-custom/gym_custom/envs/_init_.py

The custom environment file presented in Code segment 9 shows the minimal class setup. The *_init_.* method initializes the environment, the *step* method takes the an integer that represents on of all possible actions. The *reset* method is used to

reset the environment to its initial state or - depending on the goal of the agent - a random state. The *render* method is used to represent the environment, this could be a table or a GUI. When the gym environment is created, the environment can be installed using the following command "**pip install -e .**".

```
import gym
from gym import error, spaces, utils
from gym.utils import seeding

class CustomEnv(gym.Env):
  metadata = {'render.modes': ['human']}

  def __init__(self):
    ...
  def step(self, action):
    ...
  def reset(self):
    ...
  def render(self, mode='human', close=False):
    ...
```

Code segment 9: The scaffolding for a custom gym environment

Usage

So far the universal A2C agent is discussed, in this section the usage of the agent is elaborated using the product allocation problem. This includes the custom slotting environments as well as the files to train and test the agents.

Slotting environment

This section shows what the product allocation environment looks like. And how the results of the agent throughout this thesis were obtained. Three environments were created for the slotting puzzle. Version 1 is for the sequential agent whereas version 2 and 3 are for the semi- and fully-autonomous agent types. The distinction was made because the set of actions for each agent differs. The *get_score* function for the environments was the same and can be found in Code segment 10. Whereas the sequential agent needs to keep track of its last slotted location, the semi is only finished when a decision is made for each location. The fully-autonomous agent on the other hand decides when it is finished.

```
class Slotting(gym.Env):
    metadata = {'render.modes': ['human']}

    ...

    def get_score(self, show_output=False, save_score=False):
        reward = 0
        penalty = 0

        ... # Running each of the below methods to calculate the rewards and penalties.

        return float(reward - penalty)

    def check_sdf(self):
        result_1, result_2, result_3 = 0, 0, 0

        # For each product, get the locations.
        for product_index, product in self.products.iterrows():
            sdf_day_1, sdf_day_2, sdf_day_3 = product['SDF (+1)'], product['SDF (+2)'], product['SDF (+3)']
            for location_index, location in self.locations[self.locations['Slotted currently'] == product_index]
                .iterrows():
                if location['Flow rack'] and product['Units in flow rack']:
                    sdf_day_1 -= self.additions_flow_rack * product['Units in flow rack']
                    sdf_day_2 -= self.additions_flow_rack * product['Units in flow rack']
                    sdf_day_3 -= self.additions_flow_rack * product['Units in flow rack']
                else:
                    sdf_day_1 -= self.additions_pallet_location * product['Units per pallet']
                    sdf_day_2 -= self.additions_pallet_location * product['Units per pallet']
                    sdf_day_3 -= self.additions_pallet_location * product['Units per pallet']

            if sdf_day_1 <= 0:
                result_1 += 1
            if sdf_day_2 <= 0:
                result_2 += 1
            if sdf_day_3 <= 0:
```

```
                    result_3 += 1

        return result_1 , result_2 , result_3

    def check_free_locations(self):
        return self.locations['Slotted currently'].tolist().count('')

    def check_products_slotted(self):
        result_a , result_b , result_c , result_d = 0, 0, 0, 0
        for index, product in self.products.iterrows():
            if index in self.locations['Slotted currently'].tolist():
                if product['Stacking group'] == 'A':
                    result_a += 1
                elif product['Stacking group'] == 'B':
                    result_b += 1
                elif product['Stacking group'] == 'C':
                    result_c += 1
                else:
                    result_d += 1
                    print(result_a , result_b , result_c , result_d)
        return result_a , result_b , result_c , result_d

    def check_products_not_slotted(self):
        result = 0
        for index, product in self.products.iterrows():
            if index not in self.locations['Slotted currently'].tolist():
                result += 1
        return result

    def check_movements(self):
        result = 0
        for index, location in self.locations.iterrows():
            if location['Slotted initially'] != location['Slotted currently']:
                result += 1
        return result

    def check_facing_locations(self):
        result = 0
        for index, product in self.products.iterrows():
            number_of_occurrences = self.locations['Slotted currently'].tolist().count(index)

            if number_of_occurrences > 1:
                occurrences = self.locations.index[self.locations['Slotted currently'] == index].tolist()

                for o in range(len(occurrences) - 1):
                    if self.locations['Next location'][occurrences[o]] != occurrences[o + 1]:
                        result += 1
        return result

    def check_stacking_class(self):
        result = 0

        stacking_class = 0
        list_of_products_found = []

        for index, location in self.locations[self.locations['Slotted currently'] != ''].iterrows():
            if location['Slotted currently'] not in list_of_products_found:
                list_of_products_found += [location['Slotted currently']]

                current_stacking_class = self.products['Stacking class'][location['Slotted currently']]
                if current_stacking_class < stacking_class:
                    result += 1
                else:
                    stacking_class = current_stacking_class
        return result

    def check_stacking_group(self):
        result = 0

        stacking_group = 'A'
        list_of_products_found = []

        for index, location in self.locations[self.locations['Slotted currently'] != ''].iterrows():

            if location['Slotted currently'] not in list_of_products_found:
                list_of_products_found += [location['Slotted currently']]

                new_stacking_group = stacking_group
                current_stacking_group = self.products['Stacking group'][location['Slotted currently']]
                if stacking_group == 'A':
                    if current_stacking_group == 'B':
                        new_stacking_group = 'B'
                    elif current_stacking_group == 'C':
                        new_stacking_group = 'C'
                    elif current_stacking_group == 'D':
                        new_stacking_group = 'AD'
                elif stacking_group == 'AD':
                    if current_stacking_group == 'A':
                        result += 1
                    else:
                        if current_stacking_group == 'B':
                            new_stacking_group = 'B'
                        elif current_stacking_group == 'C':
                            new_stacking_group = 'C'
                elif stacking_group == 'B':
```

```
                        if current_stacking_group == 'A' :
                            result += 1
                        else :
                            if current_stacking_group == 'C' :
                                new_stacking_group = 'C'
                            elif current_stacking_group == 'D' :
                                new_stacking_group = 'BD '
                    elif stacking_group == 'BD ' :
                                if current_stacking_group == 'A' or current_stacking_group == ' B ' :
                            result += 1
                        else :
                            if current_stacking_group == 'C' :
                                new_stacking_group = 'C'
                    elif stacking_group == 'C' :
                                if current_stacking_group == 'A' or current_stacking_group == ' B ' :
                            result += 1
                        else :
                            if current_stacking_group == 'D' :
                                new_stacking_group = 'CD'
                    else :
                        if current_stacking_group != 'D' :
                            result += 1

                    stacking_group = new_stacking_group
            return result

        ...
```

Code segment 10: The reward function

Training the agent requires a Python file that initializes the model, the agent and one of the three environments. In Code segment 11 the file is depicted that was used to train the agent for the product allocation. Based on the reward function the agent got either immediate or sparse rewards.

```
...

# Agent type : sequential , semi-autonomous or fully-autonomous
agent = ' sequential '

# Reward function : sparse or immediate
reward_function = ' immediate '

# Basic variables
scenario = 'A'
updates = 100
batch_size = 10
save_model = True

# Hyperparameters
number_of_nodes = 512           # 512
hidden_layers =  0              # 0
value = 0.5                     # 0.5
entropy = 0.5                   # 0.0001
gamma = 0.95                    # 0.95
learning_rate = 0.005           # 0.0007

product_a_slotted = 15
product_b_slotted = 15
product_c_slotted = 15
product_d_slotted  =  15
free_locations = 2
matching_sdf_1 = 15
matching_sdf_2  =  10
matching_sdf_3 = 5

product_not_slotted = 20
movement = 1
facings_not_adjacent = 15
stacking_group_violation = 15
stacking_class_violation  = 10

REWARDS = {
        ' Product slotted (A) ' : product_a_slotted ,
        ...
        ' Matching SDF (+3) ' : matching_sdf_3
}

PENALTIES = {
        ' Product not slotted ' : product_not_slotted ,
        ...
        ' Stacking class violation ' : stacking_class_violation
}

MAX_ADDITIONS_PER_FLOW_RACK_PER_DAY = 2
MAX_ADDITIONS_PER_PALLET_LOCATION_PER_DAY = 7

...

locations = ... # Load locations from CSV file
products = ... # Load products from CSV file
```

```
...

# Create the s l o t t i n g environment
env = gym. make( agent_type ,
                        reward_function=reward_function ,
                        directory= directory ,
                        locations= locations ,
                        products=products ,
                        rewards=REWARDS,
                        penalties=PENALTIES ,
                        additions_ f low_ rack=MAX_ADDITIONS_PER_FLOW_RACK_PER_DAY ,
                        additions_ pallet_ location=MAX_ADDITIONS_PER_PALLET_LOCATION_PER_DAY) env

. get_score ( show_output=True )

model = Model( locations=len ( env . locations ) ,
                        products=len ( env . products ) ,
                        number_of_actions=len ( env . actions ) ,
                        number_of_nodes=number_of_nodes ,
                        number_of_extra_layers=hidden_layers )

# A2C agent
agent = A2CAgent( model , value=value , entropy=entropy , gamma=gamma, learning_ rate= learning_ rate )

rewards = agent . t r a i n ( env , updates=updates , batch_sz=batch_size )

# Saving the model weights .
i f save_model :
        model . save_weights ( directory + " /model . h5 " )

# Saving the r e s u l t s
...

agent . t e s t ( env)
env . save ( )
f i n a l_ s c o r e = env . get_score ( show_output=True )
```

Code segment 11: The script used for training the agent