SCIENTIFIC PROJECT WORK

Alexander Wuttke

Experiments with Deep-Q-Learning in a Decision Support System for Logistics Networks in Material Trading

Course	B.Sc. Mechanical Engineering
Matriculation Number	169108
Date of Submission	07.20.2018
Examiner	Univ.-Prof. Dr.-Ing. Markus Rabe
Adviser	Felix Dross, M.Sc. Software Engineering

Table of Contents

1 Introduction

Modern logistics networks represent very complex systems, which are hardly manageable without appropriate tools. Especially considering growing competition on the international markets, companies operating in them must find solutions for optimizing their logistics networks to stay competitively viable.

Decision support systems (DSSs) are a valuable tool when it comes to managing logistics networks. They are interactive, computer-based systems, which help managers or decision taking staff in their decision process by providing models, methods and relevant data (Gluchowski et al., 2008).

Over recent years there has been increasing interest in DSS in different application areas. Miller et al. (2013) created a DSS to support the daily deployment of finished goods in Pfizer's, a distributor of pharmaceuticals in the United States. Other examples are the developed strategy of Biswas et al. (2016) for multi-stage DSSs in logistics networks, aiming to optimize the distribution and transportation system, and the discrete event simulation-based DSS of Hertz et al. (2013), developed in association with five industrial partners, especially focusing on optimizing services in the industrial field.

Nevertheless, there is a high demand for the development of specialized systems, especially by large enterprises with multiple sites. The DSS that provides the framework for the experiments of this work was proposed by an international operating company and is developed by Dross and Rabe (2014; 2015). The company has more than 100 warehouses in different countries and an inventory of around 150,000 items on permanent stock, resulting in an extensive and hard to manage network. Obviously even small improvements can cause significant cost savings in this large network, making the utilization of a DSS even more worthwhile.

Rabe's and Dross's (2014) DSS uses discrete event simulation to predict the consequences of different available actions that can be applied to the logistics network, e.g. changing the stock. To provide valuable advices for the user, the DSS somehow has to learn from its simulation results and acquire the ability, to judge how promising a certain action is in a given logistics network. In their previous works, Rabe et al. (2015) propose to use reinforcement learning to make their DSS learn and they implemented a reinforcement learning agent in 2017 (Rabe et al).

The introduced reinforcement learning agent is inspired by recent research of Mnih et al. (2015), who use reinforcement learning with the Q-learning technique and deep convolution neural networks to teach their agent playing Atari 2600 games. Following the successful approach to use a deep-Q-network (DQN) agent, Rabe et al. (2017) also used the concepts of Q-learning and deep convolutional neural networks (CNNs) for their

agent. First experiences have shown promising preliminary results on small logistics networks, but more experiments have to be conducted to judge the agent's performance.

The aim of this work is to conduct and evaluate more extensive experiments using the existing framework of Rabe et al. (2017). Moreover, following their suggestion, the scaling behavior on bigger networks will be discovered as well as other CNN architectures than the one used in the first experiments. In the end, an assessment of the potential for using a DQN-agent for logistics networks in materials trading will be proposed.

To begin with, logistics networks, especially in materials trading, will be introduced and the arising challenges will be pointed out. As a possible solution approach for such challenges, DSSs are described. Special attention is paid to Rabe's and Dross's (2014) DSS as it is designed especially for logistics networks in materials trading. As their DSS uses a DQN-agent that is based on the work of Mnih et al. (2015), this work and its potentials have to be introduced. In order to understand how the results of this work are achieved, the functionality of a DQN-agent has to be explained. This includes knowledge about the concepts of reinforcement learning, Q-learning and CNNs. To understand what similarities Rabe et al. (2017) utilized to apply an agent for video games to real logistics networks their state representation has to be explained. Before describing the experiment setup, the technical implementation of Rabe et al.'s DQN-agent will be presented shortly. In the description of the experiment setup the performance indicators for the experiments are defined and the experiment environment is introduced. After the scope, purpose and environment is defined, the actual experiment results are presented. This project work ends with a conclusion of the gained results and a final assessment, whether using DQN-agents for logistics networks is a promising approach for further research.

2 Decision Support System for Logistics Networks in Materials Trading

In order to create the subject-specific basis, logistics networks, and especially logistics networks in materials trading, will be defined in the following chapters. In addition, a brief overview of DSSs for logistics networks is given.

2.1 Logistics Networks

The name logistics networks is applied for simplified models of logistics systems, created by abstraction and modelled as networks (Sucky, 2008).

Generally speaking, a system is a finite set of elements and relations between elements (Franken and Fuchs, 1974). A system in which different participants, like customers, manufacturers or forwarders, interact through a customer-supplier relationship is called a logistics system. The customer-supplier relationship describes the flow of goods, information and money between locations such as production sites, hubs or warehouses (Sucky, 2008). As a result, logistics systems can be defined as socio-technical, open, dynamic systems in which value-chains are realized (Isermann, 1998).

To analyze or simulate a real-world logistics system, the system can be modelled as a network. In the graph theory, the term network is defined as a directed and weighted graph (Jungnickel, 1994). Directed means that arcs exist, which connect two nodes of the graph with a fixed direction. Weighted means that the arcs of the directed graph have a value attached. For example, we could refer arcs to transport relations and their corresponding values to distance indications. Nodes can represent customers, which act like a sink or even more complicated facilities in which several processes are modelled.

2.2 Logistics Networks in Materials Trading

Like other logistics networks, logistics networks in material trading have to balance the logistical performance and logistical costs to reach profitability. This means that there has to be a compromise between having low stocks and having a high on-time delivery reliability, as well as having short processing times but high utilization (Wiendahl, 2014).

In materials trading a very high reliability of on-time delivery is particularly important. As materials trading most often supply essential raw materials for production lines, delays may course financial losses. This competes with low stocks. Keeping the stocks as low as possible is especially beneficial for goods of high value, such as many of the goods in materials trading (Schomberg, 2016).

Furthermore, there are especially inhomogeneous flows of goods. This is due to inconsistent demands and varying sizes and volumes of the relevant goods. Arising from

these circumstances, optimally utilizing the means of transport is a big challenge with a significant financial impact (Schomberg, 2016).

The increased complexity of logistics networks in material trading requires efficient monitoring and management, which is often opaque for the manager in charge. Therefore, the development of decision support tools is an essential matter for companies operating in this economic branch.

2.3 Decision Support System

The first concept for DSSs was designed by Keen and Morton in the 1970s. Some years after the first concept, they stated a definition in their work of 1978:

"Decision Support Systems represent a point of view on the role of the computer in the management decision-making process. Decision support implies the use of computers to:

1. Assist managers in their decision processes in semi-structured tasks.
2. Support, rather than replace, managerial judgement.
3. Improve the effectiveness of decision-making rather than its efficiency" (Keen and Morton, Decision Support Systems, p.1)

Their definition clearly points out that the computer does not take a manager's job to make decisions but eases the decision-making process.

Since the early research, DSSs evolved significantly due to technological and organizational developments. There is the development of data warehouses, online analytical processing, data mining or web-based DSSs just to name some developments from which DSSs benefited. However, those developments introduced new problems, too. For example, with the development of the Internet, logistics networks got more global, complex and connected. Moreover, the users of modern DSSs are different from users in the early 1970s, and in the course of years, the understanding and the expectations for DSSs have risen. Therefore, the development of DSSs remains an interesting field of research, which is closely connected to new technologies (Shim et al., 2002).

Dross and Rabe (2014) introduced a DSS for large logistics networks based on a Simheuristic framework. The concept of Simheuristic is extensively introduced in Dross and Rabe (2014) and can be described as a simulation-optimization approach where the evaluation function of the optimization algorithm is realized with a simulation model (Rabe et al., 2017). The work was initially proposed by an internationally operating trading company with more than 100 warehouses in different countries and an inventory of around 150,000 items on permanent stock in order to improve their network management.

The monitoring of the company's logistics network is realized with performance measurement systems, developed by experts of the company. These systems summarize multiple performance measures in key performance indicators (KPIs). KPIs are raised on a periodic basis and are sent to a responsible manager. Moreover, the systems compile lists of possible actions that can be applied to the network to improve the KPIs. An action could be the relocation of stock from one site to another or the adjustment of transport relations within the network. In the case of a deterioration of a KPI, alerts are send and the manager can apply the corrective actions. The selection of a suitable action can be difficult, due to big interdependences in the network and a wide range of possible actions. The introduced DSS targets this issue and tries to ease the decision making of the manager to help him finding a beneficial solution (Dross and Rabe, 2014; Rabe et al., 2017).

Dross uses discrete-event simulation in his work to predict the impact of actions on the network (Rabe and Dross, 2015). The discrete-event simulation is carried out by the simulation tool SimChain (Rabe et al., 2017; Gutenschwager and Alicke, 2004; SimPlan AG, 2017), which is a specialized supply chain simulation tool for Siemens Plant Simulation (Siemens PLM Software, 2017). The logistics network is represented in a MySQL database that contains all data and parameters for the simulation. The actual simulation model is instantiated from the data at runtime. To implement changes in the logistics network, e.g. applying different actions, the corresponding data in the database must be changed. After a simulation run the simulation results are stored in dedicated database tables and are available for further processing (Rabe and Dross, 2015).

The following Figure 2.1 shows how a simulation model is dynamically instantiated in Plant Simulation by combining the simulation tool SimChain and the MySQL database.

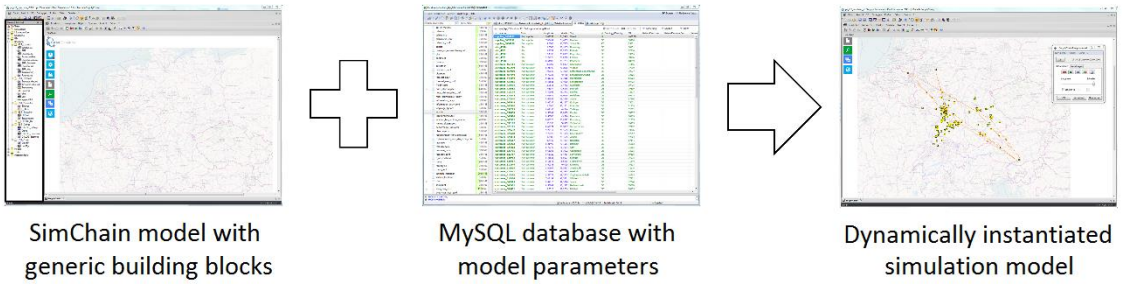

Figure 2.1: Working principle of SimChain (Rabe et al., 2017, p. 3)

Instead of judging an action's consequences by looking at the changes of the KPIs, the authors choose a simplified approach for this work's experiments, where only the accumulated costs of the network and the β service level are reference values to judge an action's impact. The β service level is a percentage measure that states how many deliveries could be provided in-time (Rabe et al., 2017).

When the DSS is used, the KPI alerts are send to a heuristic unit within the DSS instead of being directly send to the manager. The heuristic unit has the task to predict the action's possible impact and find useful action combinations. Unfortunately, not all possible actions can be tested in a reasonable time, therefore the use of a heuristic is needed. As Dross (2015) suggested in one of his works, reinforcement learning is a promising approach for the heuristic unit.

3 Realization of the Heuristic Unit with Deep Reinforcement Learning

This chapter introduces essential knowledge about machine learning and the implementation of the heuristic unit that is used for the experiments of this work.

3.1 Learning Atari-Games with Deep-Q Learning

The heuristic unit is inspired by research of Mnih et al. in 2015. Their work achieved striking results (Sutton and Barto, 2017) with the usage of deep reinforcement learning which Dross et al. (2017) refer to as a breakthrough. Mnih et al. (2015) created a single algorithm that can play and maximize the game score of classic Atari 2600 games by teaching itself with only the high-dimensional raw pixels as an input as well as the current game score. An increased difficulty arises from the different genres of the played games, reaching from side-scrolling shooters to boxing and sports games, because it is more demanding for a single algorithm to perform well in different scenarios.

The open-source framework, which enables their agent to interact with the environment, is called Arcade Learning Environment (The Arcade Learning Environment, 2018). Through this framework, the agent can utilize the 210 x 160 pixels resolution game screen with a 128-colour palette of old Atari 2600 games (Mnih et al., 2015). The home video console was published in 1977 and was very popular in subsequent years with about 30 million consoles sold (AtariAge, 2018).

Utilizing the same hyperparameters for each game, the algorithm outperformed the existing reinforcement learning algorithms at this point of time on 43 of 49 games, despite that the outperformed algorithms used more information about the played games. Even more notably, the method of Mnih et al. achieved more than 75% of the game score a professional human tester acquired in 29 of 49 games (Mnih et al., 2015).

Their algorithm uses a novel agent whom they label DQN agent. The DQN-agent combines the two concepts of Q-learning, a variant of reinforcement learning, and CNNs, which are explained in the following chapters.

3.2 The Concept of Reinforcement Learning

Machine learning can be realized by several classes of approaches. One of them is called reinforcement learning and it has created notable results recently. Besides the already mentioned work of Mnih et al. (2015), Google's project AlphaGo received a lot of attention when beating one of the best players of the very complex game Go without any handicap. Reaching this level of play was expected to be many years later according to

artificial intelligence researchers and demonstrated the potential of its usage (Sutton and Barto, 2017).

Reinforcement learning is a class of solutions for learning from interaction with an environment. The learning process is not supervised by a teacher who can judge a specific action based on experience. The algorithm has to gain its own experience and is able to research uncharted problems which makes reinforcement learning "the closest to the kind of learning that humans and other animals do" (Sutton and Barto, 2017).

Figure 3.1 visualizes how reinforcement learning works.

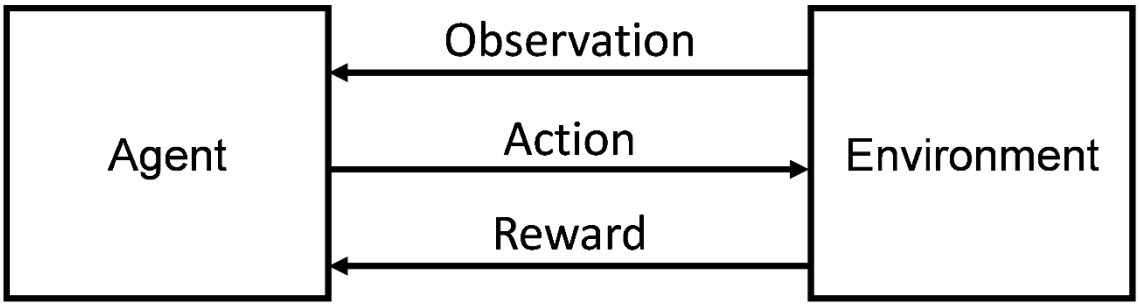

Figure 3.1: Concept of Reinforcement Learning (Rabe et al., 2017)

Reinforcement learning solutions have an agent, who can interact with an environment and has the possibility to observe the environment's current state. Based on his experience, he can perform actions in the environment and therefore has the capability to change the state of the environment. Furthermore, the agent is goal-directed, which means that the agent always considers the whole problem rather than sub-problems when selecting actions. This enables that an action might pay off later. To evaluate how the chosen action influenced the environment, the agent can get a reward. As the whole process of Reinforcement learning is closed-loop the agent repeats the steps until a predefined goal was reached or a desired number of training-steps was conducted. In order to enhance the agent's decision making, reinforcement learning aims to maximize the cumulated reward the agent gets by finding the most profit-yielding action for a specific state. This mapping of action-possibilities to states is called policy. How this policy takes place is the key difference between algorithms classified as reinforcement learning algorithms (Sutton and Barto, 2017; Goodfellow et al.,2016).

3.3 The DQN-Agent

The DQN-agent introduced in Mnih et al. (2015) uses Q-Learning, developed by Watkins (1989), to calculate values for each action in a state, which reflect how likely it is that the corresponding action yields a good reward signal in the respective state. These values for state-action pairs are called Q-values. In the course of training, the Q-values are adjusted with new experiences in order to find the optimal prediction for every state-action pair.

The function that calculates the Q-values is called Q-function. The Q-function belongs to the family of temporal-difference learning methods, which do not need to know how the environment works internally to learn. This trait allows algorithms of this family to be applied versatile for different problems. Moreover, those algorithms converge as long as the policy is fixed and enough training-steps can be carried out. Another trait of temporal-difference learning is the decision making based on experience and the partial consideration of future estimations when calculating a possibility for an action (Sutton and Barto, 2017).

Equation 3.1 shows the definition of the Q-Function, which calculates the action-value Q. Based on the Q-value that is currently calculated by the function, a new Q-value including a new experience will be calculated. On the one hand, the new learned value consists of the reward gained for executing an action, on the other hand it consists of the best Q-value selectable in the next state after executing an action, which enables the process to take further developments into account. How much influence the future best Q-value has, can be adjusted by a discount factor γ. However, the original Q-value will only be gradually approximated to the newly calculated Q-value. The learning rate defines the speed of this adjustments with a range of $0 < \alpha <= 1$, where $\alpha = 1$ would mean that the original value will be set to the new value directly.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Equation 3.1: Definition of Q-Function (Q=Q-value, S_t=current state, A_t=current action, α=learning rate, R_{t+1}=reward, γ=discount factor, maxQ(S_{t+1}, a)=best future Q-value)

However, if the agent would only select the highest Q-value available in each state while training, the agent might get stuck in his exploration process because he will only repeat the same action. Therefore, an exploration rate is introduced which will force the agent to take a random action rather than the action with the highest Q-value to ensure a constant exploration. This exploration rate varies over time and is linearly annealed, ensuring a high exploration rate in the beginning and a lower exploration rate in the end.

Some problems, such as image recognition, provide a high-dimensional input for the agent, who has to make use of old experience to recognize characteristics in new situations. As those high-dimensional inputs are most often complex, hence have many states, the agent cannot learn for every possible state explicitly as it would consume too much resources. Therefore, the agent tries to generalize the given input and learns actions for patterns found in the input. One technique of doing so is the CNN, a method affected by how a human's brain accomplishes early image recognition (LeCun et al., 1998). In terms of Q-learning, a CNN can be used as a function approximator of the Q-function to map actions to constellations of patterns in a state (Mnih et al. 2013).

CNNs built of multiple hierarchically arranged layers of different functionality and increasing level of abstract are called deep CNNs (LeCun et al., 2015). The output of one layer becomes the input of the following layer, an architecture called feedforward. One of these layer classes is the convolutional layer which consists of multiple feature maps. These maps can be seen as neurons, that indicates where certain features, like edges or curves, are in the input matrix. Features are described by small matrices, referred to as kernels, and can be detected in the input-matrix by the mathematical operation of convolution. In this operation, the kernel is multiplied with a part of the input-matrix of the same size and the higher the resulting value the more the investigated part, also called a receptive field, matches the feature. To create the feature map, the same kernel is slid over the input-matrix with a set stride and convolution is used. Due to the usage of the same kernel in a feature map, convolutional layers have the trait of equivariance to translation, meaning that if features are translated in the input data it will still be detected but translated in the output data, too. However, convolutional layers are not capable of detecting scaling or rotation of features (Sutton and Barto, 2017; Goodfellow et al., 2016).

As the operation of convolution is a linear operation, the network is linear itself, as long as only convolution is used. Therefore, changing the parameters can only result in a solution that is a linear function of its input and any other solution cannot be achieved. That is why after computing the feature map, a non-linear activation-function has to be applied to the feature map's result. In modern CNNs most often rectified linear units are used. This is a simple non-linear activation function $g(z) = \max\{0,z\}$ which changes all negative values in a convolutional layer's output to zero and has proven to be the best choice in terms of training time (Goodfellow et al. 2016).

Another layer for building CNNs is the fully connected layer that consists of neurons, which have full connection to all activations in the previous layer. The output of a fully connected layer is calculated by a matrix multiplication and the addition of a bias offset. A fully connected layer is always the last layer of a CNN. Typically, the last layer has as much neurons as there are available performable actions. In the case of the DQN-agent, the outcome of the last fully connected layer is the function approximation of the Q-function (Sutton and Baro, 2017).

In order to improve the CNNs results, training steps have to be conducted. The agent learns from experiences, which consist of 4 components: An initial state, the action that was carried out, the corresponding reward and the new state, after executing the action.

When a Q-value was calculated for an experience, and the calculated value differs from the given approximation of the CNN, the CNN has to lower or optimally eliminate the error. The error between the computed value and the real value can be expressed by a cost-function and minimizing this function makes the CNN more accurate. The

minimization can be achieved by optimization methods, premising that the gradient of the function is known. An algorithm to compute the gradient of a CNN's loss function with respect to the parameters is the back-propagation algorithm which uses backward propagation of information through the network (Sutton and Barto, 2017) and the recursive application of the chain rule (Goodfellow et al., 2016; LeCun et al., 2015).

If the gradient of the loss function is computed, gradient-based optimization methods are used to perform the actual learning. The gradient indicates how the weights of each layer have to be changed and depending on the learning rate, the weights of the CNN's layers are shifted to the indicated values (LeCun et al., 2015).

The gradient-based optimization method used in the DQN-agent is the RMSProp algorithm. The algorithm uses an adaptive learning that is scaled inversely proportional to an exponentially decaying average. It causes to discard experience from the past so it can rapidly converge after finding a convex bowl in the function to minimize. It has proven to be one of the most favored optimization methods regarding deep neural networks and its behavior is defined by a momentum and a minimum square gradient (Goodfellow et al., 2016).

While training, Mnih et al. (2015) also rely on two further techniques. The first technique is called experience replay, it pools experience transitions over time and provides the agent minibatches, containing multiple randomly selected transitions to learn from, rather than learning just from the most recent experience. This reduces correlations between experiences and allows experiences to be used in several training steps, ensuring a more efficient data usage. Another advantage is the reduction of the risk of being stuck in a local minimum or even diverge. The second technique also aims to improve Q-learning's stability by using a separate network, called the target network, which is copied in a constant interval for calculating the Q-values from, resulting in less likely oscillations and divergence.

3.4 State Representation and Rewards for the Experiment Environment

A good quality of the state representation supports the successful use of CNNs (Sutton and Barto, 2017). The state representation used in this work is described in Rabe et al. (2017). The aim was to create an image as an input based on Mnih et al. (2015). The illustrated state representation consists of two types of segments which are joined together and added up to an image as seen in Figure 3.2.

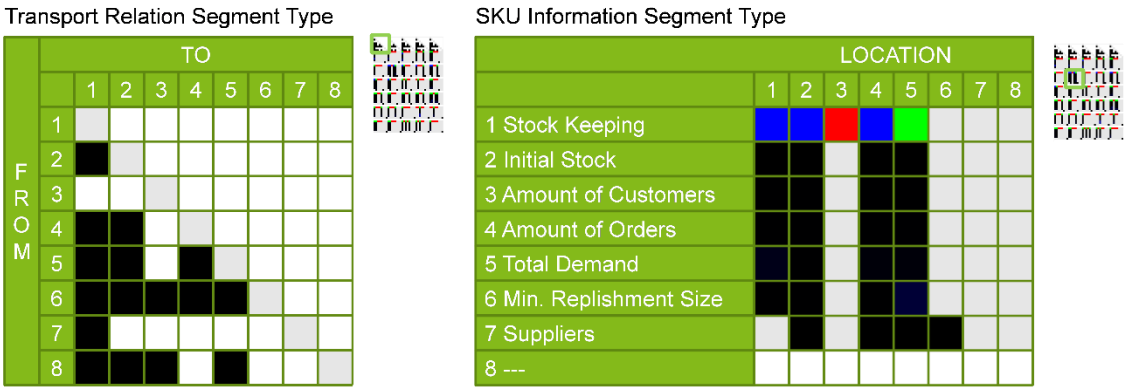

Figure 3.2: State Representation of a Logistics Network (Rabe et al., 2017, p.8)

One segment type represents the transport relations in the logistics network and is constructed like a distance matrix. The other segment type holds information about all used stock keeping units (SKUs). Values are stated by different RGB color values, resulting in a 3-dimensional matrix. While the transport matrix is represented by the two colors white and black (is connected or not), information about SKUs are illustrated by the whole color spectrum. The size of all segments is the same, as well as their position in different states, so the information about a specific SKU will always be found on the same position. This allows the CNN to spot features in the state representation easier. Furthermore, scaling the state representation for different sized models is no problem since only more segments have to be generated (Rabe et al., 2017).

The reward signal for the agent is calculated from the change of the network's overall logistics costs and β service level. The β service level is a performance indicator expressing the share of on-time deliveries compared to the number of total deliveries (Rabe et al., 2017). If the overall costs could be decreased, a positive reward is granted, otherwise it is negative. The β service level's percental change is multiplied by a constant factor and summarized with the overall cost's reward. With the constant factor, the impact of the change of β service level can be adjusted. Before the reward signal is passed to the agent, the summarized reward has to be scaled. Mnih et al. (2015) used a clipping of 1 for positive rewards and -1 for negative ones. Rabe et al. (2017) use a scaling between -1 and 1, to better reflect how impactful an action is. The agent can also suggest an action, which cannot be performed on the current state. The reward for those not executable actions is set to -1.

The general working principle of the combination of discrete-event simulation and the DQN-agent by Rabe et al. (2017) is shown in Figure 3.3.

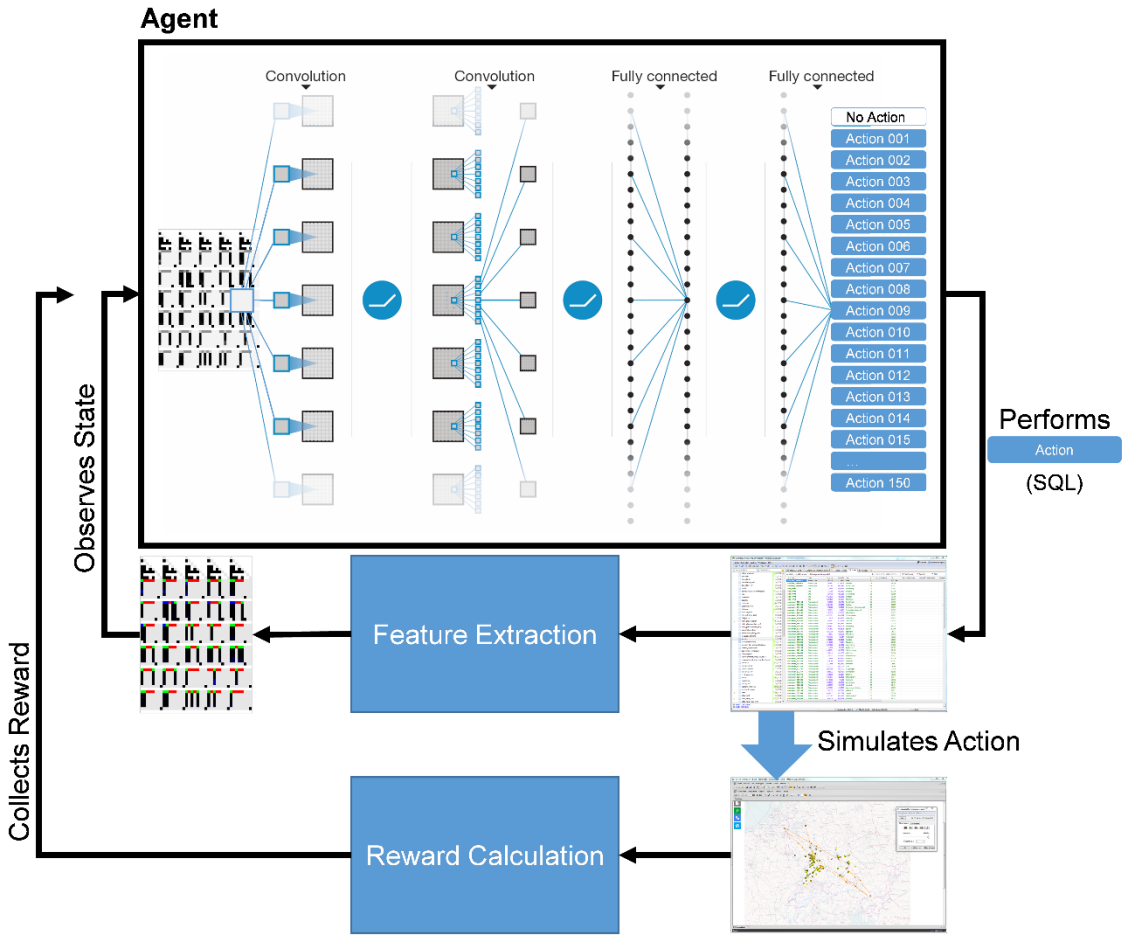

Figure 3.3: General Working Principle of the Combination of Discrete-Event Simulation and the DQN-Agent (Rabe et al., 2017, p. 6)

3.5 Implementing Machine Learning with Tensorflow

To implement the DQN-agent, Rabe et al. (2017) use a powerful tool for implementing machine learning projects called Tensorflow (Tensorflow, 2018). Tensorflow is an open source library developed in the course of the "Google Brain project" in 2011. The library helps conducting largescale machine learning and deep neural network research on several platforms like personal computers or mobile devices running on Android or iOS. It has been used in several research projects and commercial products and is expected to impact a wide field of domains (Goodfellow et al., 2016).

A computation in Tensorflow is realized by a directed graph, consisting of nodes which model built-in operations like additions, matrix multiplications or even more complicated computations as gradient computations. Between connections of those nodes flow tensors which transport the output of one node to the input of another node. Usually the whole graph is defined before the first computation takes place and executions of the full graph or just parts of it, called subgraphs, are carried out in the course of events. Tensorflow provides two frontend languages, C++ and Python (Abadi et al., 2015).

After learning about DQN-agents and the implementation in this project work, the author's experiments and their environment are introduced in the next chapters. Moreover, the results of the experiments will be presented and evaluated.

4 Experiments with Reinforcement Learning

As the preliminary results of Dross et al. (2017) have shown, the utilized agent is able to improve small logistics networks. The more extensive experiments in this work will investigate how the agent copes with bigger networks in terms of cost improvement, β service level, runtime and scaling behavior. Dross et al. (2017) used the same CNN architecture as Mnih et al. (2015) in their experiments, but as they propose in their conclusion, due to having a different problem at hand, another architecture might achieve better results. Therefore, different architectures will be tested.

4.1 The Simulation Models

The experiments of this work will be conducted on a small example data set of Dross's (2014) logistics network to keep simulation-times low. Because the scaling will also be investigated, two versions with different scopes are created. In one version, 30 SKUs are considered, the other version features 60 SKUs. Both networks contain three suppliers and five sites. The suppliers provide SKUs for the sites, from which the SKUs are transported to the customers. The smaller model (30 SKUs) consists of 103 customers and processes 176 orders, the bigger model (60 SKUs) consists of 147 customers and 275 orders.

For this project work, two types of actions can be applied to the model. One action causes a stock keeping site to no longer keep a SKU on stock and therefore receive it from another site on demand. As this action can be applied to all sites and all SKUs in a model, it results in 150 possible actions for the smaller and 300 actions for the bigger model. However, not all actions can be executed in every state. For example, an action cannot be executed two times in a row because the SKU is not kept on stock already after the first execution and is therefore invalid. This will result in a negative reward. Additionally, the agent has access to one more action type, the "no action"-action, to enable the system to stay in its current state.

4.2 Architectures for the Experiment Environment

Mnih et al. (2015) used a CNN with four hidden layers in their work. The first three layers are convolutional layers with the following rectified linear units. The first layer convolves 32 filters of 8 x 8 with a stride of four. The second one has 64 filters of 4 x 4 with stride of two and the third one has 64 filters, too, but of size 3 x 3 and with a stride of one. The final hidden layer is a fully connected layer with 512 rectifier units.

The agent's success depends very much on the provided state representation. Following the different nature and state representation of the problem at hand, adjusting the CNN might increase the agent's efficiency. In photos or screenshots of video games, the pixels

are graphical representations of logical game objects and are most often arranged in clusters, so neighboring pixels are much likely to be interdependent. Furthermore, game objects itself are clusters in the image, too, and form features in features. In contrast to this, Rabe et al.'s (2017) state representation doesn't use such high interdependencies of neighboring pixels and features in features often. Following this, the use of a shallower CNN with only one convolutional layer and one fully connected layer is suggested by the author.

Therefore, two additional experiments will be run on two shallower architectures shown in Table 4.1. When it comes to choosing parameters for a CNN, a good performing architecture is identified by testing several architectures and comparing their performance. In this work, only 2 additional architectures will be tested and compared as testing more would be beyond this work's scope. Both CNNs use a convolutional layer with filters of size 8x8 and a stride of 8. By choosing the filter size and stride according to the segment size in the state representation the CNN examines all information of a whole segment at once. The difference between the architectures is the number of filters in the convolutional layer, one possesses 512 and the other one 1024.This number is much higher than in the architecture of Mnih et al. (2015) and is chosen to cope with a high number of features in a segment. The number of features in the state representation of Rabe et al. (2017) is comparably higher because of the lesser interdependencies of pixels in a segment, which requires more identified features for good performances. Additionally, the number of neurons in the fully connected layer varies, too, as the architecture with more filters in its convolutional layer has only 512 neurons, the other architecture has 1024 neurons. Doubling the number of neurons in one architecture compared to the architecture of Mnih et al. (2015) helps to differ the identified features for a state even more and might produces even better results.

	Architecture II	**Architecture III**
CL Size	8 x 8	8 x 8
CL Stride	8	8
CL Number of Filters	512	1024
FC Number of Neurons	1024	512

Table 4.1: Parameters for Architecture II and Architecture III

4.3 Experiments with Deep-Q-Learning

The first experiments were conducted with the same architecture (Architecture I) and training parameters as used by Mnih et al. (2015) for their agent to play Atari video games.

Because their training parameters are made for problems with much more simulation steps than the problem at hand, two minor changes had to be done. This concerns the "target network update frequency" and the "final exploration after episode" hyperparameters, as the target network would not get enough updates within the executed simulation steps and the final exploration rate would never be met. The hyperparameters for the first experiment are shown in table 4.2. An explanation of the parameters is given in chapter 3 as well as in Mnih et al. (2015).

HYPERPARAMETER	VALUE	DESCRIPTION
Minibatch size	32	Number of transitions over which training is computed.
Replay memory size	1.000.000	Number of most recent transitions
Target network update frequency	100	Frequency of target network updates
Discount factor	0.99	Discount factor ɣ used for Q-learning
Learning rate	0.00025	Learning rate used by RMSProp
Momentum	0.95	Gradient momentum used by RMSProp
Minimum squared gradient	0.01	Squared gradient momentum used by RMSProp
Initial exploration rate	1	Initial greediness
Final exploration rate	0.1	Final greediness
Final exploration after episode	3000	Number of episodes after reaching the final exploration rate

Table 4.2: Training Parameters I

In order to judge the agent's success, the reward he gets for proposing actions will be constantly monitored. Other performance indicators are the change of costs and β service level. Based on the change of these performance indicators it can be easily shown whether the agent improved the logistics network or not. When the β service level increases and the costs decrease, the logistics network is improved by the agent. When only one

indicator changes in the desired direction, so the costs decrease or the β service level increases, it is important how much the values changed and how much their deviation is weighted compared to each other. That means, if there is only a small loss in the β service level but high cost savings, overall the logistics network might be improved on basis of the user's criteria. The reward includes this weight and therefore a positive reward will always indicate an improved logistics network while a negative reward will always indicate a deterioration. A reward of zero will indicate no changes to the performance and costs of the logistics network.

The results of each experiment will be visualized in two line graphs. One chart shows the development of the average reward in evaluation runs the agent got in the course of an experiment. The horizontal axis on the graph shows the point of time in episodes. It reaches from 0 to 5000 episodes as this is the observed number of episodes in an experiment run. The vertical axis shows the achieved average reward in an evaluation run for the given time. The value range is from the maximal reward of 1 to the minimum reward of -1. Moreover, a logarithmical trendline is applied to indicate to which value the average reward will converge. The second line graph shows the course of the β service level change as well as the change of costs. The horizontal axis is the same as in the first graph, the vertical axis now shows the values of the β service level change in percent and of the cost change in Euro. However, the cost change must be scaled down to fit the same range as the β service level change by a factor of 10000.

Figure 4.1 shows the graph of the average reward in the evaluation runs of the experiment. For any given evaluation run, the average reward is negative, which indicates a poor learn effect of the agent and it shows no tendency to improve over time. Accordingly, the agent is not able to improve a logistics network with the same hyperparameters as for video games in the examined period. This outcome is underlined by the mainly negative influences on the β service level and the particularly low cost savings as shown in Figure 4.2. However, despite punctually improving the logistics network for example at episode 3750, where one can see an increased β service level and lower costs, the reward is negative. This is due to the behavior of rewards, which punishes the selection of actions not being available in the current state with a negative reward. Remarkable is that the best average rewards were produced in the beginning, which points out the instability and scarce learning ability.

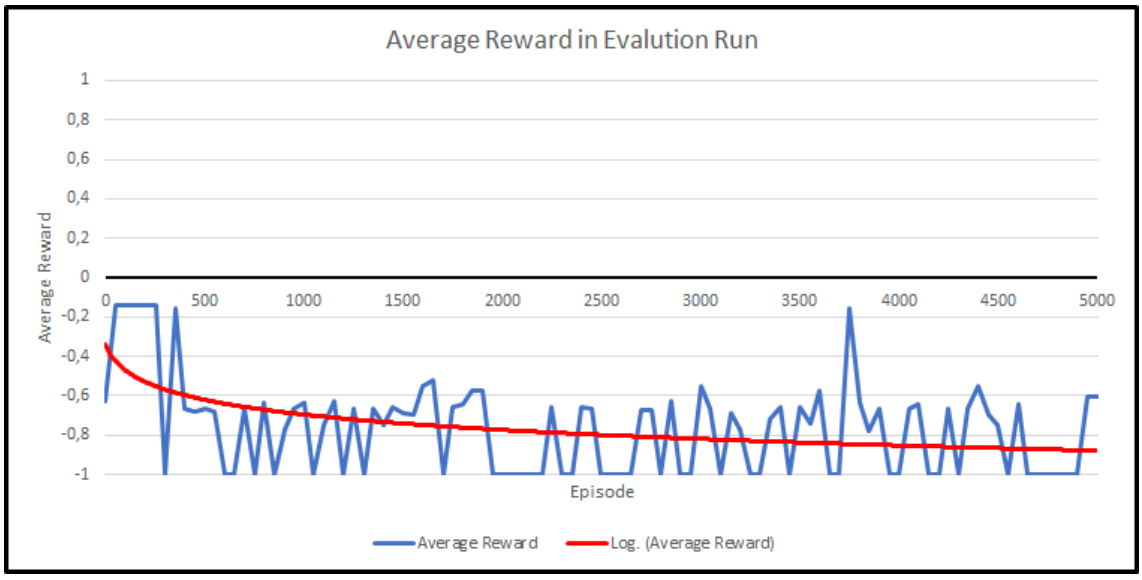

Figure 4.1: Average Reward of an Experiment with Architecture I and Training Parameters I on a 30 SKU Logistics Network

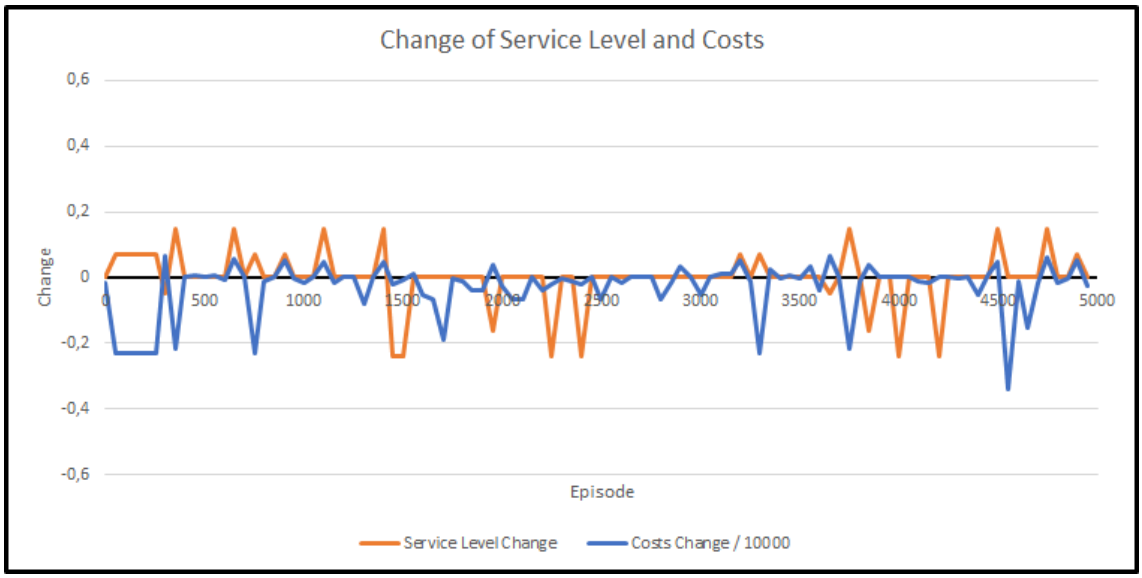

Figure 4.2: Change of β Service Level and Costs of an Experiment with Architecture I and Training Parameters I on a 30 SKU Logistics Network

The bad results may depend on an inappropriate architecture, a wrong selection of training parameters or both. To proof, whether Architecture I can be successfully used for other problems, too, a different set of hyperparameters for training is tested (Table 4.3).

Hyperparameter	Value
Minibatch size	500
Replay memory size	1000000
Target network update frequency	20
Discount factor	0.1
Learning rate	0.000025
Momentum	0.95
Minimum squared gradient	0.01
Initial exploration	1
Final exploration	0.3
Final exploration after episode	1000

Table 4.3: Training Parameters II

Figure 4.3 visualizes the average reward in all evaluation runs for Architecture I and parameters of Table 4.3. The average reward is significantly higher than in the first experiment, yet it is still on the negative side for most of the time. Besides a local maximum at about 3000 episodes, the results of the last 500 episodes state a learn effect over time.

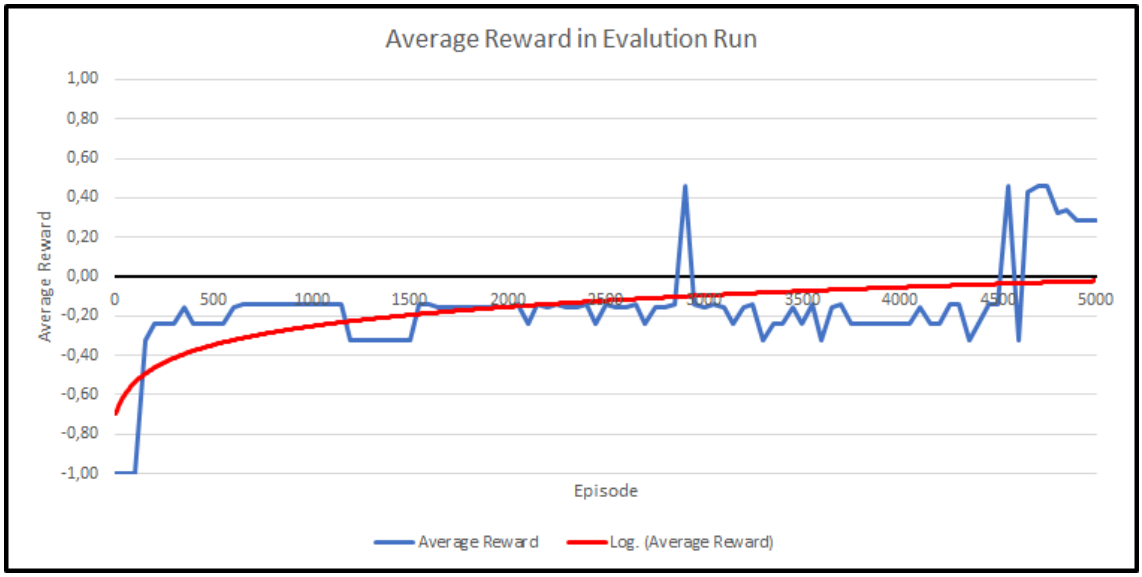

Figure 4.3: Average Reward of an Experiment with Architecture I and Training Parameters II on a 30 SKU Logistics Network

Figure 4.4 clearly shows the agent's ability to improve a logistics network at hand. In every evaluation run, the agent was able improve or at least retain the β service level, a cost reduction could be achieved over the whole observed period.

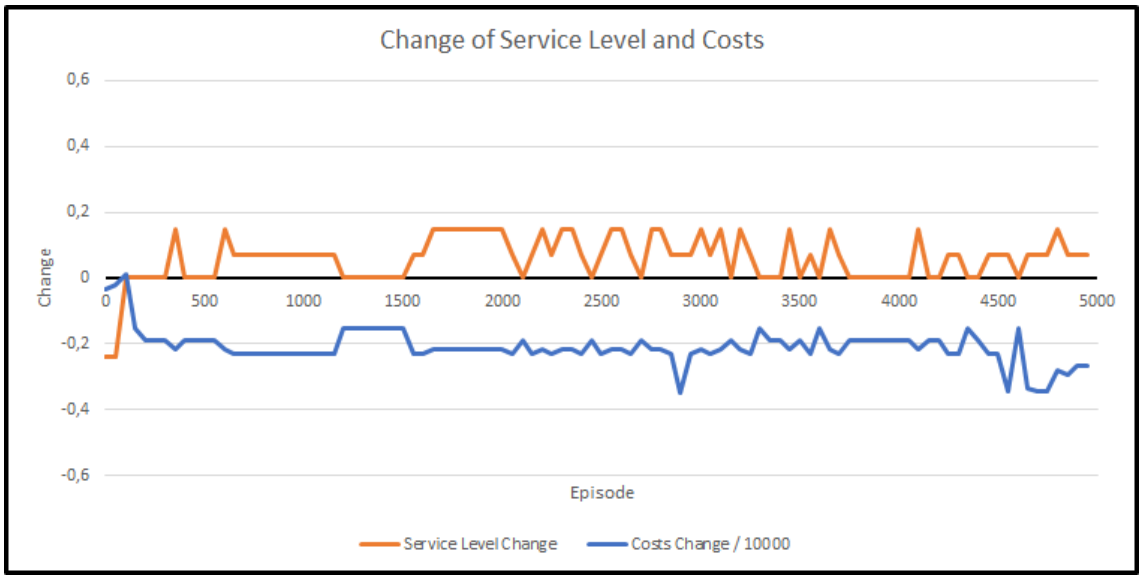

Figure 4.4: Change of β Service Level and Costs of an Experiment with Architecture I and Training Parameters II on a 30 SKU Logistics Network

Nevertheless, the results are not striking in terms of the yield average reward and the study of other architectures might reveal better options to improve the quality of the network even more. The experiment with Architecture II provide better results, as the average award quickly turns into positive values and at about half of the episodes, constantly staying in very high regions and still improving a little bit in the end as Figure 4.5 indicates. Compared to the results of Figure 4.4, Figure 4.6 only show slightly more cost savings while the β service level remains at the same value.

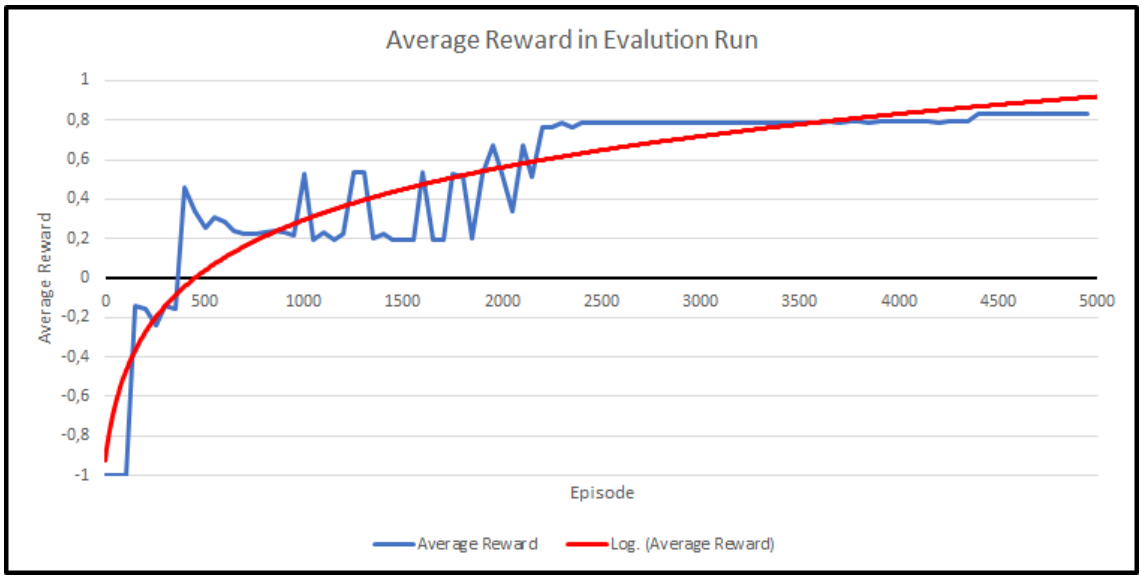

Figure 4.5: Average Reward of an Experiment with Architecture II and Training Parameters II on a 30 SKU Logistics Network

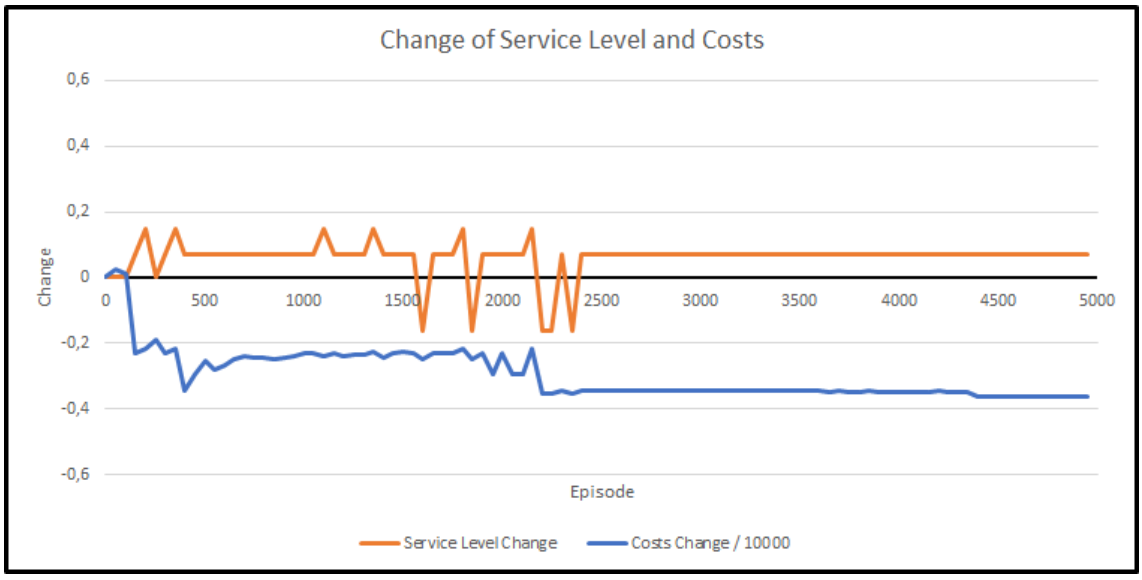

Figure 4.6: Change of β Service Level and Costs of an Experiment with Architecture II and Training Parameters II on a 30 SKU Logistics Network

Nevertheless, the average reward of this experiment is much higher than the previous one, which is explainable when considering the high penalty costs for invalid actions. Therefore, Architecture II performs remarkably better in avoiding high negative rewards in a reasonable time.

Another experiment with the same training parameters but using Architecture III produces even better results. High average reward values are achieved even more quickly and the reached maximum average reward is higher than in the earlier experiments (Figure 4.7, Figure 4.8)

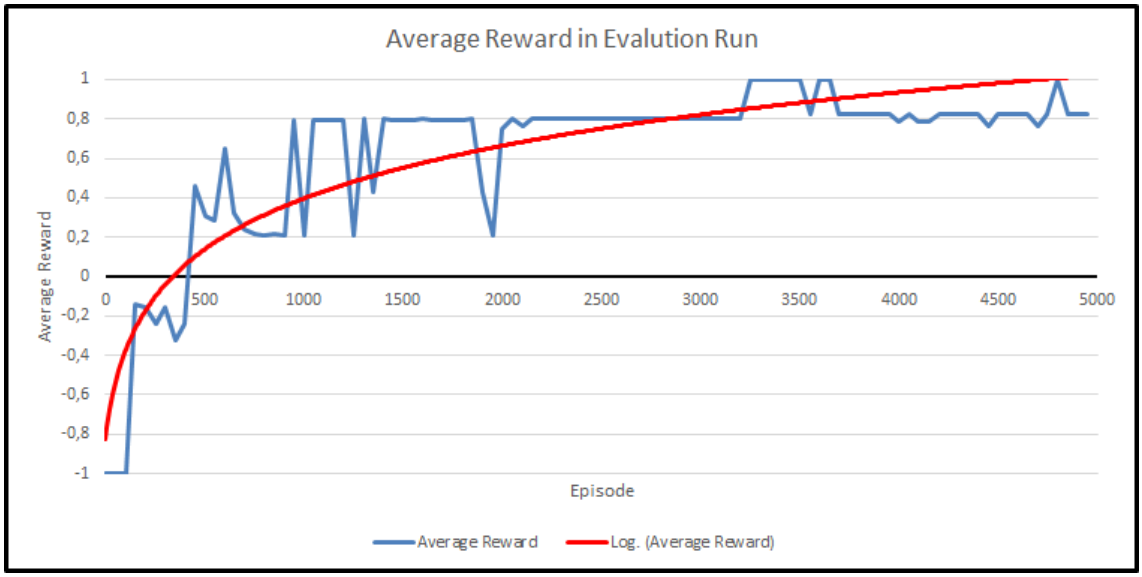

Figure 4.7: Average Reward of an Experiment with Architecture III and Training Parameters II on a 30 SKU Logistics Network

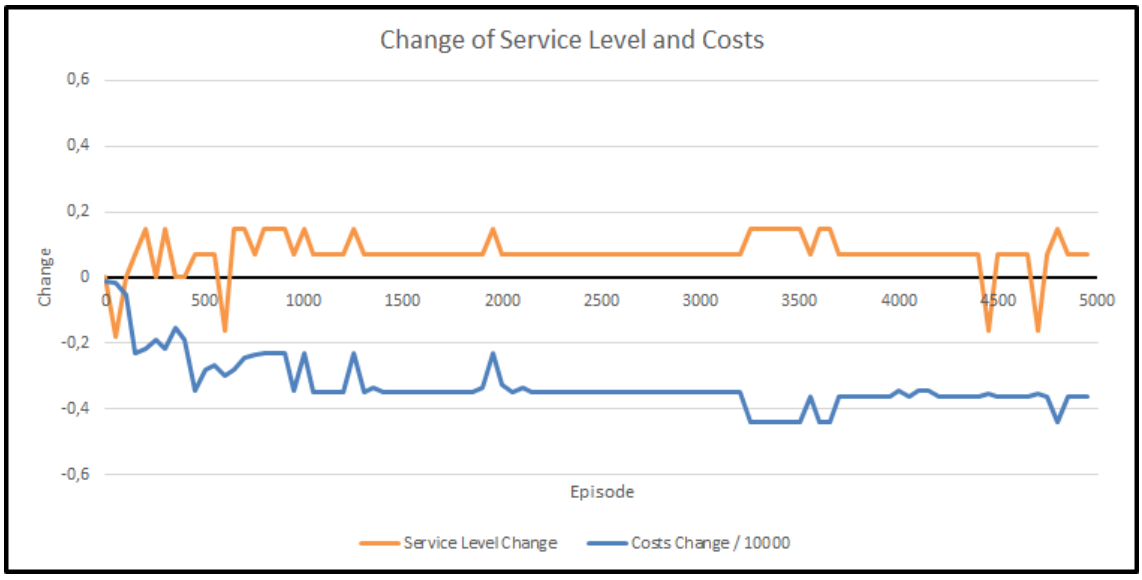

Figure 4.8: Change of β Service Level and Costs of an Experiment with Architecture III and Training Parameters II on a 30 SKU Logistics Network

The first experiments on the small 30 SKU logistics network have shown that Architecture III produced the most promising results, followed by Architecture II and Architecture I. To see how well the introduced architectures cope with larger networks, experiments on a 60 SKU logistics network are conducted, too. The set of training hyperparameters for those experiments are still Training Parameters II. The following Figure 4.9 and Figure 4.10 shows the results for the architecture of Mnih et al. (2015).

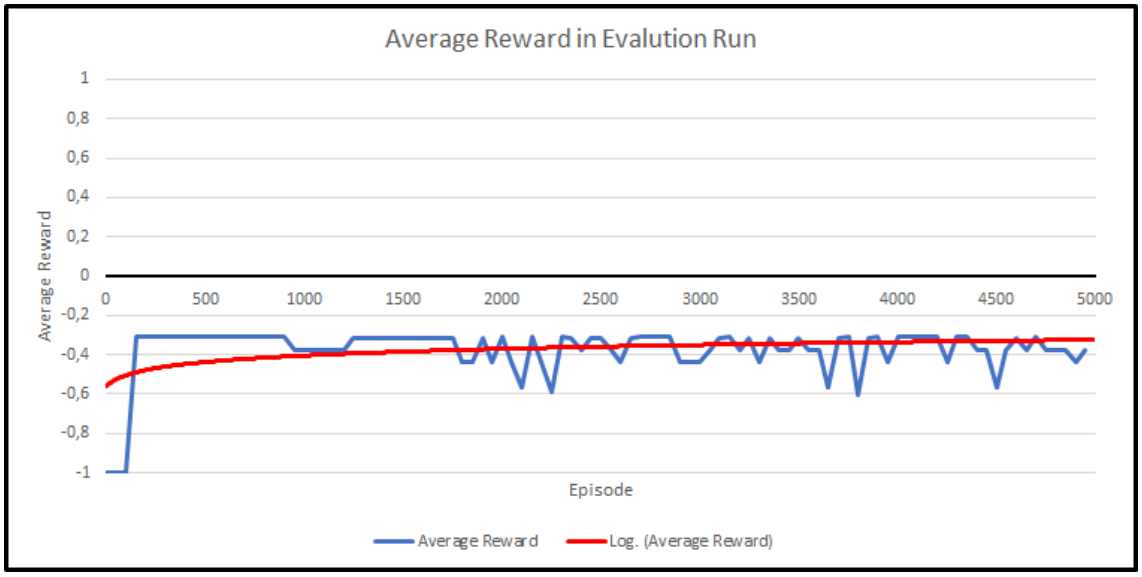

Figure 4.9: Average Reward of an Experiment with Architecture I and Training Parameters II on a 60 SKU Logistics Network

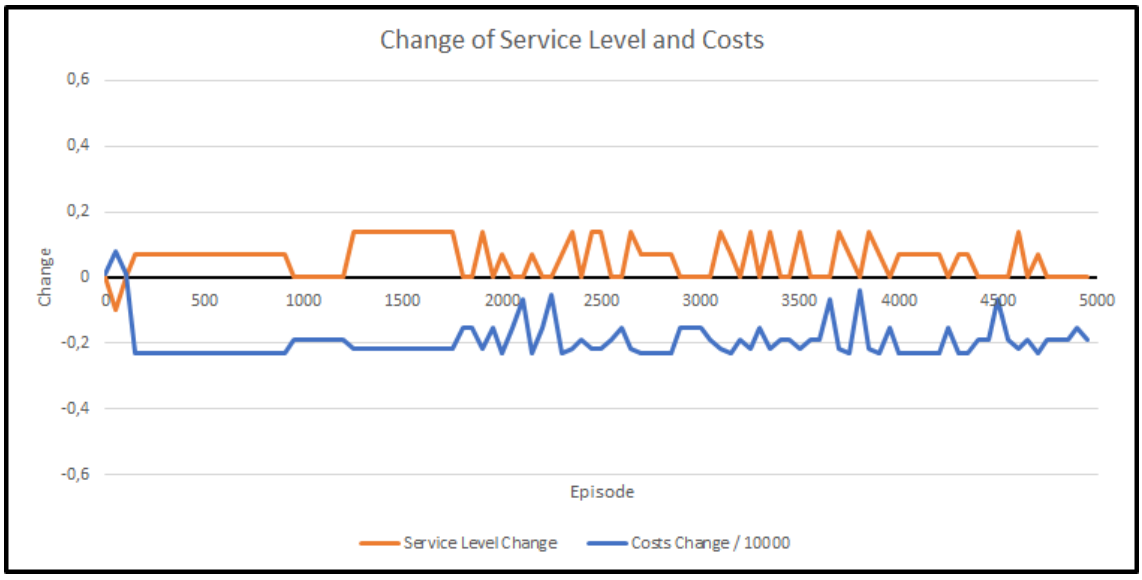

Figure 4.10: Change of β Service Level and Costs of an Experiment with Architecture I and Training Parameters II on a 60 SKU Logistics Network

The results presented in Figure 4.10 show that the agent can improve even a little larger logistics network by either increasing the β service level or lowering the overall costs. However, the average reward of the evaluation runs, as shown in Figure 4.9, is slightly worse than in the experiment with Architecture I on the small logistics network.

On the larger logistics network, the experiment with Architecture II show a mostly similar pattern. Positive average rewards are attained relatively quick and after a short period of fluctuating values, the average reward is constant and improves a little bite over time. However, the obtained average reward is a little bit lower than on the small logistics network (Figure 4.11, Figure 4.12).

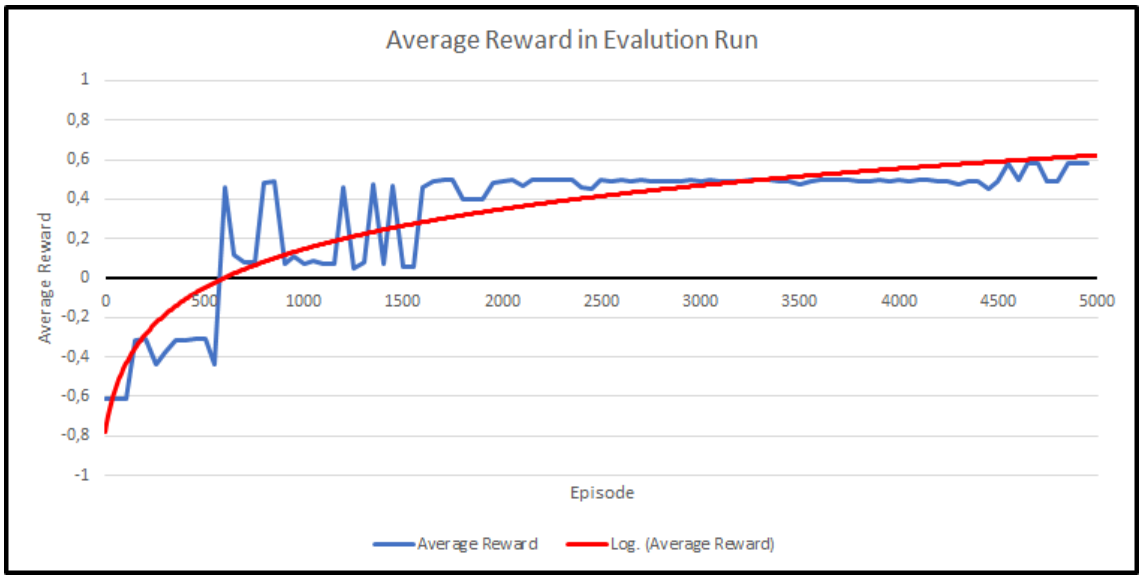

Figure 4.11: Average Reward of an Experiment with Architecture II and Training Parameters II on a 60 SKU Logistics Network

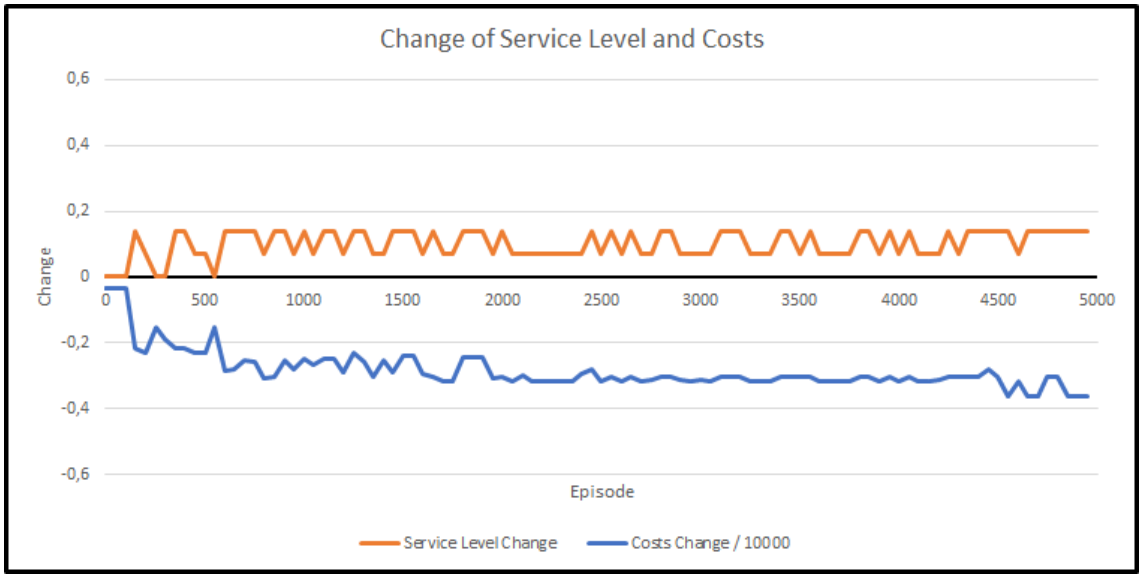

Figure 4.12: Change of β Service Level and Costs of an Experiment with Architecture II and Training Parameters II on a 60 SKU Logistics Network

Figure 4.13 and Figure 4.14 visualize the results of the experiment with Architecture III. Conspicuous is the volatile course of the graph, which seems to steady in the end. Unfortunately, the observed number of periods is not sufficient to tell if this trend continues.

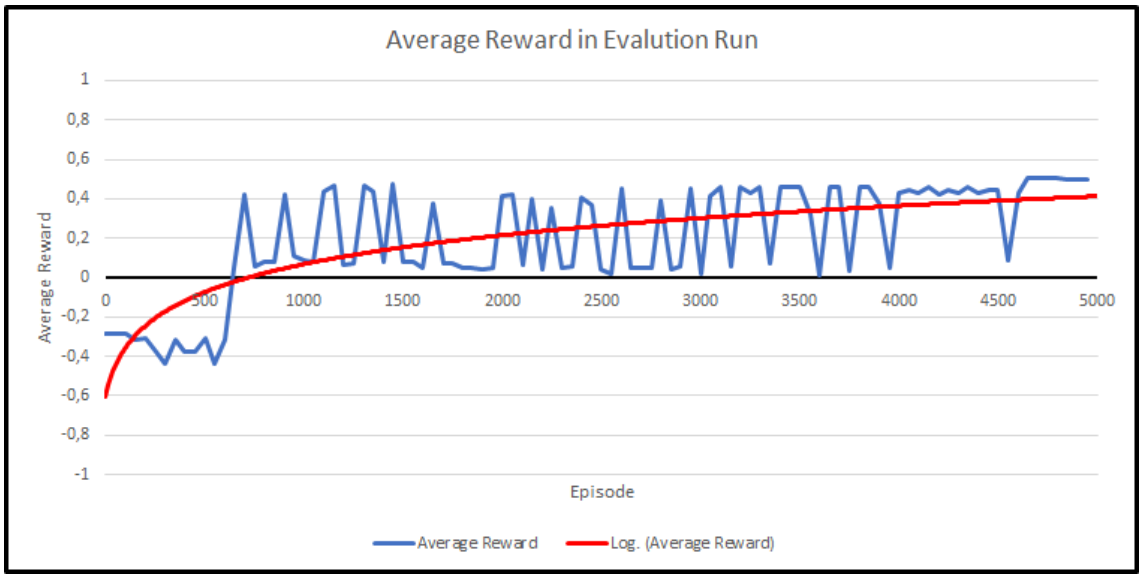

Figure 4.13: Average Reward of an Experiment with Architecture III and Training Parameters II on a 60 SKU Logistics Network

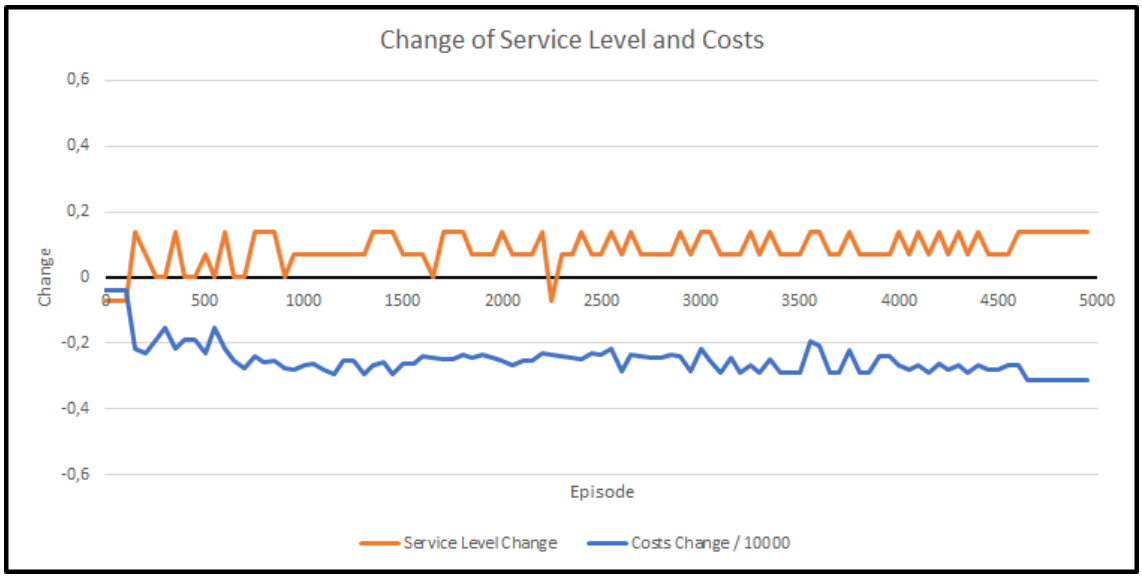

Figure 4.14: Change of β Service Level and Costs of an Experiment with Architecture III and Training Parameters II on a 60 SKU Logistics Network

The total running times of all conducted experiments are stated in Table 4.4.

	30 SKU	**60 SKU**
Architecture I	39h 34min	63h 03min
Architecture II	52h 46min	74h 16min
Architecture III	51h 36min	78h 36min

Table 4.4: Experiment running times

The measured time is the time period from the first conducted training step is until the last evaluation run has finished, therefore the given time includes the needed time for the training as well as all evaluation runs. Architecture I is less time consuming than its shallower counterparts, despite having a more complex structure. As the number of neurons and filters in architecture I is much lower, the faster computation time is easily explainable. The running times of Architecture II and Architecture III do not differ much in both cases but seems to be a little bit higher when using Architecture II on the small model but lower on the larger model. This observation is consistent with the performance level, so the better performing architecture is less time. However, this is not expected when considering that a better performing architecture most likely chooses more executable, valid actions to avoid negative rewards and therefore has to carry out more simulation steps than an architecture which chooses many invalid actions and can skip the corresponding simulation.

Table 4.4 shows that the training takes at least some days to achieve stable results, for convergence even more time will be needed. Therefore, when using the algorithm at hand in a DSS, the system has to be trained several days before it can be used. Lowering the

number of filters in the convolutional layer of Architecture II and Architecture III might decrease the computing time as the higher number of filters seem to be the biggest factor for the high computing times compared to Architecture I.

5 Conclusion and Outlook

The results have clearly shown that the investigated DQN-agent of Rabe et al. (2017) is able to improve logistics networks in materials trading. For the logistics networks used in this project work, large cost savings and β service level improvements could be achieved. When extending a model, much more time is needed for the same number of episodes. However, the agent's performance level does not drop significantly, so even larger logistics networks will be examinable.

Following the positive results of the experiments, the utilization of a DQN-agent for logistics networks in materials trading has the potential to achieve good results on larger networks. Moreover, it could be observed that shallower CNNs achieve better results than the deep CNN of Mnih et al. (2015). In this work two different architectures for a shallow CNN were tested. It seems that the number of neurons and number of filters have an influence on the scaling behavior of logistics networks. But the adaption of them could improve the performance and calculation time even more. The author recommends to proceed future research in this direction.

References

AtariAge: Atari 2600 History. http://www.atariage.com/2600/ (Accessed 17.04.2018).

Dross, F.; Rabe, M.: A SimHeuristic Framework as a Decision Support System for Large Logistics Networks With Complex KPIs. In: Wittmann, J; Deatcu, C. eds.: *Proceedings of the 22nd Symposium Simulationstechnik (ASIM 2014).* Berlin, Germany: HTW Berlin, 2014, pp. 247–254.

Franken, R.; Fuchs, H.: Grundbegriffe zur Allgemeinen Systemtheorie. Grochla, E.; Fuchs, H.; Lehmann, H. eds.: *Systemtheorie und Betrieb*. zfbf Sonderheft, 1974, pp.23–50.

Gluchowski, P.; Gabriel, R.; Dittmar, C.: Management Support Systeme und Business Intelligence. Computergestützte Informationssysteme für Fach- und Führungskräfte, 2nd ed.. Berlin, Heidelberg: Springer, 2008.

Goodfellow, I.; Bengio, Y.; Courville, A.: Deep Learning (1. ed). Cambridge, Massachusetts, USA: MIT Press, 2016.

Gutenschwager, K.; Alicke, K.: Supply Chain Simulation mit ICON-SimChain. In Spengler, T.; Voß, S.; Kopfer, H. eds.: *Logistik Management. Prozesse, Systeme, Ausbildung.* Heidelberg: Physica-Verlag HD, 2014, pp. 161–178.

Hertz, P.; Cavalieri, S.; Finke, G. R.; Duchi, A.; Schönsleben, P.: A Simulation-Based Decision Support System for Industrial Field Service Network planning. In: *Simulation: Transactions of the Society for Modeling and Simulation International*, Vol. 90 (1). London, England: SAGE Publications, 2013, pp. 69–84.

Isermann, H.: Grundlagen eines systemorientierten Logistikmanagements. In: Isermann, H. ed.: *Logistik - Gestaltung von Logistiksystemen, 2. Aufl.* Landsberg: Moderne Industrie, 1998, pp.21–60.

Jungnickel, D.: Graphen, Netzwerke und Algorithmen, 3rd ed.. Mannheim: BI-Wissenschaftsverlag, 1994.

LeCun, Y.; Bengio, Y.; Hinton, G.: *Deep Learning*. In: *Nature*, Vol. 521. England: 2015.

Miller, T.; Peters, E.; Gupta, V.; Bode, O.: A Logistics Deployment Decision Support System at Pfizer. In: Furman, K.; Chaovalitwongse, W. eds.: *Annals of Operations Research*, Vol. 203 (1). Boston: Springer US, 2013, pp.81–99.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. NIPS Deep Learning Workshop, 2013.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; Hassabis, D.: Human-level Control Through Deep Reinforcement Learning. In: *Nature*, Vol. 518. England: 2015, pp.529–533.

Rabe, M.; Dross, F.: A Reinforcement Learning Approach for a Decision Support System for Logistics Networks. In Yilmaz, L. ed.: *Proceedings of the 2015 Winter Simulation Conference*. Huntington Beach, CA, USA: Omnipress, 2015, pp. 2020-2032.

Rabe, M.; Dross, F.; Wuttke, A.: Combining a Discrete-event Simulation Model of a Logistics Network with Deep Reinforcement Learning. In: *Proceedings of the MIC and MAEB 2017 Conferences*. Barcelona, Spain: 2017, pp. 765-774.

Schomberg, L.: Anwendungskonzept für ein logistisches Assistenzsystem im Werkstoffhandel. 2016.

Shim, J.P.; Warkentin, M.; Courtney, J.F.; Power, D.J.; Sharda, R.; Carlsson, C.: Past, Present, and Future of Decision Support Technology. In: *Decision Support Systems*, Vol. 33(2). Amsterdam: Elsevier, 2002, pp. 111-126.

Siemens PLM Software: Tecnomatix Plant Simulation. http://www.plm.automation.siemens.com/de_de/products/tecnomatix/plant_design/plant_simulation.shtml. (Accessed 8 September 2017).

SimPlan AG: SimChain. http://www.simchain.net. (Accessed 8 September 2017).

Sucky, E.: Netzwerkmanagement. In Arnold, D.; Isermann, H.; Kuhn, A.; Tempelmeier, H.; Furmans, K. eds.: *Handbuch Logistik*. Berlin: Springer, 2008, pp.934–945.

Sutton, R.S.; Barto, A.G.: Reinforcement Learning: An Introduction. Cambridge, Massachusetts; London, England: MIT Press, 2017

Samanta, S.; Biswas, T.: A Strategic Decision Support System for Logistics and Supply Chain Network Design. In: *Sadhana*, Vol. 41(6). New Delhi, India: Springer India, 2016, pp. 583-588.

Tensorflow https://www.tensorflow.org/ (Accessed 17.04.2018)

Watkins, C.J.C.H.: Learning from Delayed Rewards. Cambridge University, 1989.

Wiendahl, H.P.: Betriebsorganisation für Ingenieure. Mit 3 Tabellen, 8th ed. Munich, Germany: Hanser, 2014.

List of Abbreviations

CNN	Convolutional Neural Network
DSS	Decision Support System
DQN	Deep-Q-Network
KPI	Key Performance Indicator
SKU	Stock Keeping Unit

List of Figures

List of Tables

List of Equations

Eidesstattliche Versicherung
(Affidavit)

Name, Vorname
(Last name, first name)

Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der Bachelor-/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Ort, Datum
(Place, date)

Unterschrift
(Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Ort, Datum
(Place, date)

Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.