Technische Universität München
Fakultät für Maschinenwesen
Lehrstuhl für Automatisierung und Informationssysteme
Univ.-Prof. Dr.-Ing. Birgit Vogel-Heuser

Bachelor's Thesis

CONCEPT AND IMPLEMENTATION OF AN AI-BASED PRODUCTION-MANAGEMENT APPROACH FOR A CELL-BASED MANUFACTURING PROCESS

Konzept und Implementierung einer KI-basierten Produktionssteuerung für eine zellbasierte Fertigung

Author:	JAN NALIVAIKA
Student-ID:	03694590
Supervisor:	Prof. Dr.-Ing. Birgit Vogel-Heuser
Advisor:	Iris Weiß, M.Sc.
Industry-Advisor:	Dr.-Ing. Peter Robl
	Oliver Lohse, M.Sc.
Issue date:	May 6, 2020
Submission date:	November 4, 2020

Acknowledgments

My special thanks go to Dr.-Ing. Peter Robl, head of the T REE MDM DMT-DE department of Siemens AG, as well as Oliver Lohse, for their competent support and for the time they have taken for me and my work. Their expertise was invaluable in formulating the research question. I would particularly like to thank my advisor Iris Weiß for her dedicated support and guidance. Her professional advice, insightful and constructive feedback and made this work possible.

Abstract

In today's production systems, one-off productions and individual products are becoming the new regular and are no longer a rarity. The linear conveyor belt production, which is designed for mass production, is more and more replaced by a cell-based production, which can meet every customer's individual requirements. A specially adapted schedule is required for a profitable manufacturing process, which optimally utilizes the manufacturing machines' potential to achieve maximum profit. Traditional scheduling algorithms are not considering unexpected events, such as machine breakdowns. This thesis aims to create a system capable of reacting to changing situations on the shop-floor like machine breakdowns and adapting the schedule accordingly. This reaction must happen without time-delays to not impact the manufacturing process any further. The created system must determine how to reprioritize products and process steps to keep the manufacturing process profitable by maintaining a high throughput of products. This goal is realized with the implementation of a Reinforcement Learning algorithm. The presented method shows what is required to implement a Reinforcement-Learning-approach for online-scheduling in a cell-based manufacturing process and how the necessary elements interact with each other. The results show that the trained algorithm is capable of the desired online-reaction and achieves significantly better results than a FIFO scheduling rule. This thesis provides a valid proof of concept that can be developed further to control a physical manufacturing process in a real-world application.

Table of Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Description
AI	Artificial Intelligence
APS	Advanced Planning and Scheduling
ERP	Enterprise-Resource-Planning
FIFO	First-in-first-out
MDP	Markov Decision Process
ML	Machine Learning
NN	Neural Network
PS	Process step
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RWT	Remaining working time
SM	Scours machine
TM	Target machine

1 Introduction

1.1 Motivation

The continually changing manufacturing environment, e.g. customer demands and new manufacturing processes, requires companies to continually adapt to not be squeezed out of the market. Competitiveness is greatly affected by the company's ability to quickly and efficiently adopt new technologies that can be used to improve the performance of manufacturing systems [1]. Traditional software applications such as scheduling algorithms are limited in their reaction speed and are therefore a possible limiting factor for a manufacturing plant that has to adapt the manufacturing order in case of an unforeseen event [2]. Artificial intelligence (AI) has the potential to provide companies with new chances and yet undiscovered possibilities to further increase productivity [3]. In the last years, AI has left the developmental state and is spreading rapidly into all industry sectors [4][5]. Making use of the decreasing prices of computing power, AI made significant advances that can be utilized in a multitude of cases [4]. The ability of AI to solve real-world problems is shown on multiple occasions [6][7]. Especially in the area of manufacturing, AI is gaining much importance over recent years [8]. Modern manufacturing-processes are highly dynamic environments, producing a variety of customized products with strict deadlines [9]. The ability to adhere to the dedicated time-frame is mainly impacted by machine-failures. These machine failures are known but unpredictable and unavoidable [10]. Depending on the issue, it can take days to repair a faulted machine. For cost-efficient manufacturing, data from the manufacturing process must be available and evaluated instantaneously. On this basis, decisions like rerouting, rescheduling or re-prioritization have to be made as quickly as possible. The use of AI can meet these requirements [5]. Traditional models for decision-making can no longer process the growing volume of data over time [11]. The possibilities to process complex information from a dynamic environment will increase significantly in the future with the use of AI-based systems. The employment and further development of AI is not only crucial for manufacturing processes but beneficial for the entire mechanical engineering sector [8]. The problem that many companies still face to this day is that the implementation of the desired AI capabilities

into the current manufacturing-systems is only possible if certain conditions such as sensorization, data collection and data processing are fulfilled [12]. Due to the continuous change in the industrial environment, the contexts of the "Industrie 4.0" initiative and the increasingly close relationship between physical and digital processes, a basis is being created that will enable the promising use of AI methods to increase productivity [5][13]. Among other things, it is the goal to use pattern and relationship recognition to find undiscovered correlations that can lead to higher throughput and thus to improved production performance [14]. AI can significantly increase the rate of production and adherence to deadlines by recognizing those complex interconnections [15]. There are already classical software algorithms that can be coupled with existing Enterprise-Resource-Planning-systems (ERP-system) that offer possibilities to make well-founded decisions in the case of unexpected events, but are not suitable for usage in real-world manufacturing environments, as many aspects of physical manufacturing processes, such as transportation times between the individual machines, are not taken into account [1][16]. Smart factories are designed to take patterns, relations, and real-time events into consideration and solve problems like resource allocation and scheduling [17]. By exploiting data from all available sources, the use of AI has the potential to react to changing situations promptly and to adjust the manufacturing process accordingly, in order to reduce idle times and thus to achieve the maximum profits [9]. In situations where traditional scheduling algorithms reach their limits due to the amount of data to process, AI offers reliable production-control alternatives that, for example, help to complete production orders before the deadline and reduce the overall costs. The optimal use of multi-skill manufacturing machines, combined with the optimal priority of manufacturing steps, reduces idle times of machines and thus increases the competitive advantage [14].

1.2 Problem Formulation

In a manufacturing process, faults such as machine failures or unplanned lack of personnel have a negative impact on time constraints for the following products, which require one or more process steps at the unoccupied or faulted machine [18]. Due to the delays that occur, for example, delivery dates cannot be met and the required minimum throughput for profitable manufacturing is not achieved. Such events contribute to rising processing times and inevitably lead to cost increases and profit loss in the manufacturing process [19].

There is no system in place that is able to quickly react to the current state of production, and reroute products to alternative machines or rearrange the order of process steps to minimize the effects of the disruption incidents [20]. Traditional scheduling algorithms that are incorporated in Advanced-Planning and Scheduling-systems (APS-system) do not have the possibility to immediately make informed decisions, based on information from the current manufacturing process, on how to rearrange the manufacturing sequence of the products and process steps [21]. No solution is provided that can react close to real-time to changing situations on the shopfloor [22]. Foresighted planning of manufacturing steps in the event of a fault is impossible in a highly dynamic environment of individual products with increasing quantities [23]. Traditional scheduling algorithms are only designed to create optimal production-orders in a static environment, without unforeseen events [18]. Optimized schedules are losing their advantage due to unexpected changes on the shop-floor [24] [9] [25]. The incorporation of those dynamic events is a requirement to guarantee the promised delivery date and product quality [20]. There is currently no concept in production-control or fault management that is able to quickly react to these unforeseen events, with different degrees of influence on the manufacturing process, and draw the best possible conclusions in such a dynamic environment.

By evaluating the data collected from manufacturing processes, manufacturing companies hope to gain information that can be used to achieve a significant increase in productivity [13] [22]. Due to the ever-shortening product life cycles and the associated increase in individualized product orders, the collected information is becoming more and more varied and diverse [26]. For optimal use of the data, the weighting, and thus the relevance of the data plays an important and essential role. The weighting is the evaluation of information concerning its importance and its influence on the production process [27]. A representation of this is, for example, the downtime of a machine or the average occupancy time and its effect on the deviation from the planned production end-dates. Which data points are required for this purpose and how they have to be prepared to serve as a source of information to improve the manufacturing process, has not yet been clearly defined in any generally applicable context

[28]. When manually adjusting the order of manufacturing steps from intuition and experience based on the locally available data, it is not possible to make optimal decisions [5]. This problem is due to the non-obvious correlations, such as the transport times between machines or the individual failure probabilities of the production participants [29] [27]. The correlation of available resources and their interdependencies is impossible to grasp for a human being, especially when presented with large amounts of interconnected and unsorted data [30]. The success of a manual adjustment of the production sequence depends on the consideration of the dependencies existing in production [4].

1.3 Aim of this Thesis

This Bachelor's Thesis aims to develop a concept for a production-management approach that is able to reduce arising costs caused by unexpected machine failures. The goal is to reduce processing times by rerouting products to alternative machines and therefore reduce idle times of functional machines. The result of selecting alternative machines for certain process steps has to significantly reduce process times without resulting in any further time delays for the manufacturing process.

The conceptional system must be able to react to the changing state of the manufacturing process, especially to unexpected and unpredictable machine failures, and return instructions for actions that reduce the effect of machine failures. These changes in the production schedule have to be calculated instantaneously without leading to further time-delays for the manufacturing process. For this, special attention is given to the online creation of schedules and prioritization of products in case of a malfunction.

To accomplish this optimization, the information from the manufacturing process that is required is identified. It is explained how this information has to be processed and presented to achieve the requested increase of the production rate in manufacturing, in the case of unexpected occurrences.

1.4 Requirements for the developed Method

After the aim of the thesis is defined, the different requirements (R1-R3) for the solution and the constraints (C1-C6) for the used simulation are determined. The concluded requirements and constraints are listed in table 1.1.

A Reinforcement Learning algorithm (R1) is selected as the sole decision-making instance to ensure an online-reaction capability (R2) that makes it possible to react to changing situations on the shop-floor without time delays. The online-reaction capability is required to not further impact the timing in the manufacturing process in a negative way. Calculated decisions must be available instantaneously to reduce further idle times. The Reinforcement Learning algorithm is responsible for calculating optimal decisions to achieve the defined goal in every situation. For a simple implementation that does not require multiple lines of communication between the algorithm and the simulation where the algorithm is trained, a single-agent setup (R3) is required. This means that only one communication line is implemented between the algorithm and the simulation. The simulation provides the agent with the necessary information and the algorithm returns its decisions.

A cell-based manufacturing environment (C1) is required to create an environment that is capable of reducing the negative effects of unexpected events by the rerouting of products. These events are machine breakdowns (C4) and occur randomly. Machine breakdowns are the most important physical incidents in the manufacturing process and thus must be considered by the decision-making algorithm. The individual cells can have multiple skills (C2) that allow them to perform the necessary process steps of rerouted products. To mimic a real-world manufacturing environment more closely, the additional skills of machines vary in efficiency (C3). Only one machine in the simulated environment is capable of performing one process step the most efficiently. Other machines are less efficient, in other words, they require more time for the same process step. The products move independently of each other (C5) in the manufacturing process to enable the most freedom possible. This freedom of movement allows for a more adapted schedule that is not possible by following a predefined heuristic. The last element that has to be considered by the simulation, as well as the algorithm, is the time it takes one product to move from one manufacturing cell to the other. This time is regarded as transportation time (C6) and should not be disregarded.

Table 1.1: Requirements for the developed Method and Constraints for the used Simulation

Abbreviation	Requirement
R1	Reinforcement Learning
R2	Single-Agent setup
R3	Online-Action capability
C1	Cell-based manufacturing environment
C2	Individual cells can have multiple skills
C3	Cells can perform process steps with different efficiencies
C4	Consideration of Machine breakdowns
C5	Independent movement of products
C6	Consideration of transportation-time

2 State of Research and Development

This chapter gives an overview of multiple published approaches for implementing AI in the manufacturing process. For this, the necessary technical terms are defined and their field of application and function is presented.

2.1 Routing, Scheduling and Fault-Management

In the following, technical terms from the manufacturing process are explained. It is explained how a production order is scheduled and how unexpected disruptions are handled.

The **routing** is a sequence of the required process steps that is determined on the basis of drawings and bill of materials [31]. Routing contains the information about the necessary operations that have to be performed and the machines on which the specific product has to be manufactured is described. The routing also contains the required times necessary for the execution of the process steps [32]. In most application cases today, routing does not contain alternative routs that describe how to manufacture the product in another sequence [33].

Routing is the basis for **scheduling**. During production-scheduling with the help of an APS-system, the prioritized or only available routing variant is extended with current and forecasted information [33]. The aim of scheduling is to determine the starting-time and required machine for each process step. Its main purpose is to ensure that the capacities of the plant, including machines and personnel, are utilized to the fullest potential and a schedule of the individual manufacturing steps for production is made [32]. The result of the scheduled process steps are the production orders. The production orders contain the individual process steps for each part in the routing and additionally, their scheduled start and finish times. Further contents of a process step are the processing time on the machine, the required personnel time, the execution time, the transportation time as well as the set-up time [33].

Online-Scheduling, also referred to as Dynamic-Scheduling, is the approach of scheduling that takes real-time events into account to generate a suitable schedule for every situation as quickly as possible without any time delays. This functionality is not yet implemented into the available APS-systems. [25][10]

If faults occur along the manufacturing process, or if an order with a higher priority has to be carried out that is not scheduled, the previously calculated schedule of the production orders become invalid [22]. The calculated starting times for the process steps can not be met and a solution is required to mitigate the forming delays or staff shortages. In this situation, the production supervisor is responsible for **fault-management** and decides spontaneously how process steps and scheduling should be adjusted. As a result, no information about the changing situation is available at the production management level about what, where and how much is currently being produced. The manual adjustments are not regarding the global state of production, like machine availability, and thus might negatively influence further production. The interconnected information for an optimal adjustment can not be understood by a human [30]. Information about the current state of production is needed for scheduling future production orders [13]. In the event of a machine-malfunction, rescheduling is not possible on the APS level due to the time-consuming and sometimes non-existing feedback of the current state of production orders in production. Furthermore, rescheduling is not desired because, in this case, products that are not affected by disturbances are affected by rescheduling. This decreases stability in production significantly [34].

2.2 Usage of Production-Control-Systems

For a profitable production, that optimally utilizes the available resources, a production-planning and scheduling-system is integrated and used for the manufacturing and assembly process. How to efficiently utilize all resources for manufacturing is a long-discussed problem in the manufacturing-area [35]. Production scheduling belongs to the class of Nondeterministic Polynomial-time hard problems, which are very hard and time-consuming to solve [2] [36]. It therefore requires time and specially designed algorithms to produce adequate results [37] [38]. It is necessary to take all constraints in the manufacturing process, like efficiencies of multi-skilled machines, into account to create a schedule that allows for maximum performance [39]. To realize this, ERP and APS-systems are used for central production planning. ERP systems are business-management software tools that support a variety of operational and planning business processes of a manufacturing company. Production-planning activities belong to the basic functions of an ERP-system [40]. The range of functions of ERP-systems depends strongly on the needs of the company in terms of its IT- and organizational-structure

[41]. Planning activities in the ERP-system works in the range of days up to months and can only very vaguely estimate the given situation on the manufacturing date backlogs, machine condition and rush orders. For this reason, production planning by ERP-systems is limited to a rough schedule that assumes unlimited resources and capacities [40]. ERP-systems meet business management requirements and support production processes, but not to the extent necessary for close to real-time scheduling of manufacturing processes. ERP-systems have a database in which the required data for the manufacturing process is stored and which enables it to operate independently [42]. A piece of important information in the ERP-system is the routing, which defines how a product can be produced, independent of time and machine [43]. The current and forecasted production orders, shift schedules and existing capacities must be taken into account, to form a manufacturing schedule [40].

2.3 Definitions to the Topic of Artificial Intelligence

The following section defines the most important technical terms on the topic of AI and presents their field of application.

Artificial Intelligence (AI) is defined by John McCarthy as "the science and engineering of making intelligent machines". This field of research in computer science deals with the development of intelligent and autonomous systems that can take action based on a defined base of priorities. [44]

"**Machine learning** (ML) is a subdivision of AI specifically focused on how a computer system is able to solve a given task" [45]. ML is an over-category for a multitude of approaches that are designed to recognize patterns and regularities on the basis of available data sets and to develop solutions for presented problems [46]. Subdivisions of ML are, for example, Neural Networks, Genetic algorithms or Inductive learning [15]. The main purpose of ML is to give computer systems the ability to learn autonomously [4].

Training in the context of ML is defined as the process of improving on the task of the algorithm. By repeatedly performing the task and receiving information on its success, parameters inside the algorithm are adjusted [47] [46]. The process of adjusting the parameters is similar in all subdivisions of Neural Networks and is called backpropagation [48].

"A **neural network** (NN) is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the interunit connection strengths, or weights, obtained by a process

of adaptation to, or learning from, a set of training patterns" [49]. The parallel arrangement of processing elements, also referred to as **neurons**, is called a **layer** [49]. The category of NN is further subdivided into Supervised Learning, Unsupervised Learning and Reinforcement Learning [15].

"**Supervised Learning** are the various algorithms generating a function that maps inputs to desired outputs. One standard formulation of the Supervised-Learning task is the classification problem: the learner is required to learn (to approximate the behavior of) a function which maps a vector into one of several classes by looking at several input-output examples of the function" [47]. The name of this subdivision of NNs is derived from the fact that the input-data is labeled. The available labels of the data make it possible to correct the mapping of the algorithm, in other words it is "supervising" the performance of the algorithm. By giving feedback to the algorithm, its mapping abilities are improved [4].

"**Unsupervised Learning**, is the task of finding structures hidden in collections of unlabeled data without [the] supervision [of a human or computer]" [46]. The goal is to identify how a typical data group looks like [50].

"**Activation functions** are functions used in neural networks to computes the weighted sum of input and biases, of which is used to decide if a neuron can be fired or not." [51]. The activation functions can influence the required number of layers and the number of neurons in a layer. The most common activation functions are "hyperbolic tangent" (sigmoidal) and "rectified linear unit" (ReLU). These functions transform the input of a neuron to a new value and forward it to the next neurons. [52]

"**Reinforcement Learning** (RL) is an area of Artificial Intelligence; it has emerged as an effective tool towards building artificially intelligent systems and solving sequential decision making problems" [53]. By repeatedly performing chosen actions, the algorithm is reinforcing its knowledge on how to act in certain situations [46]. This method is mimicking the learning process of humans and animals [4].

2.4 Classification of Artificial Intelligence

An overview of the available ML-learning methods is presented and the distinct differences are laid out for a clear understanding. Supervised Learning and Unsupervised Learning are briefly discussed before the in-depth discussion of RL. This is important to understand why RL is chosen for scheduling in a dynamic manufacturing environment.

2.4.1 Advantage of ML

AI is a relatively new sector in manufacturing that can bring a lot of value in the coming years. Online-Scheduling (see Chapter 2.1) is one of the most interesting and rapidly developing areas of application of AI in manufacturing [54]. This rising interest is possible due to the increasing calculating power of computers and their falling prices [4]. Especially with the development of powerful Graphics Processing Units (GPUs) and specialized open-scours libraries, the computation power is increasing significantly [55] [56]. AI is a wide variety of techniques to solve different kinds of problems. It ranges from simple decision trees to complex NNs [15]. In this thesis, the focus lies on a very specific section in the field of AI to apply to the presented problem. The range of application of all subsections of AI is very wide and will not be discussed in this Thesis in detail. The most important field of AI in the case of this Thesis is the subsection of ML. Most ML-methods rely on NNs [4].

A NN in computers-science is a way to imitate the structure and function of actual biological neurons that can change their behavior by adapting specific values that define how they react to inputs [57]. The clear advantage of ML in comparison to rule-based algorithms is that problems can be solved without being specifically told on how to solve the problem [4]. Another point is that outputs can be generated promptly that could be considered almost in real-time and the performance is improved over many training episodes [13]. One of the disadvantages of ML is that it is not possible to transfer the knowledge from one task to another. For every specific problem, time-intensive retraining is necessary [58]. Figure 2.1 shows that the field of ML is subdivided into multiple categories.
In this work, the focus is on the subdivision of NNs. The subdivision of NNs can be further divided into Supervised and Unsupervised Learning as well as Reinforcement Learning (RL). The main criterion of ML is the ability to learn from experience [4]. An easy way to implement experience is the creation of a lookup-table. Figure 2.2 shows how a lookup-table is designed.

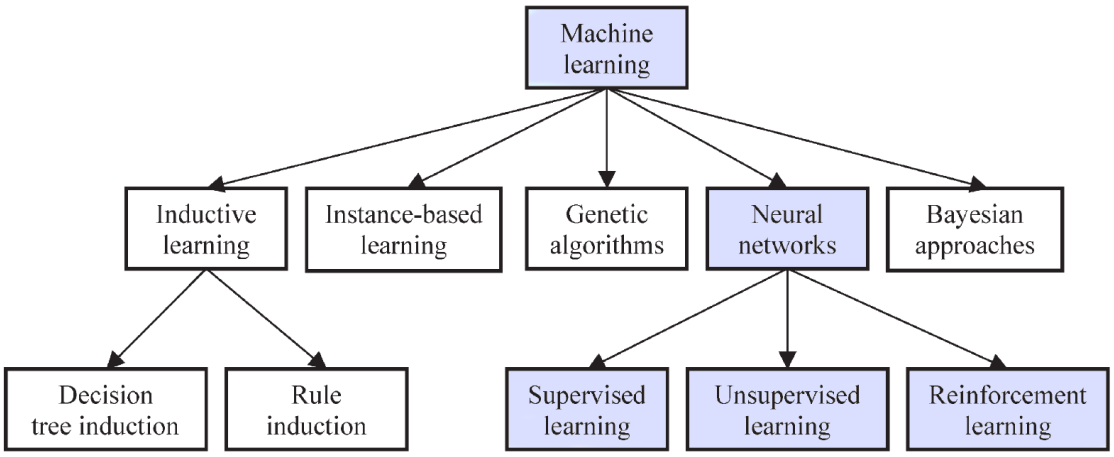


Figure 2.1: Subsections of ML [15]

If : Rain, Wind, Temperature below 0	Stay at home
If : Rain, no Wind, Temperature below 0	Stay at home
If : Sun, Wind, Temperature below 0	Go outside
If : Rain, Wind, Temperature above 0	Stay at home
If : Sun, Wind, Temperature above 0	Go outside
If : Sun, no Wind, Temperature above 0	Go outside
...	...

Figure 2.2: Example of a Lookup-Table

This table contains information about what exactly to do in which situation. This approach quickly runs into the problem of scalability. Only a certain amount of knowledge can be stored in lookup-tables this way. NNs are used to replace these large lookup-tables [46].

The knowledge of the NN is stored with the help of parameters that define the connection of neurons in the net. These parameters are called weights. The main purpose is to decrease the computational burden of updating large sections of lookup-tables [22] [46].
ML is mandatory for autonomous systems to optimize their behavior over time through experience and to acquire additional skills for new tasks [59] [15].

2.4.2 Function and Application of Supervised Learning

Supervised Learning is the process of computerized learning with a data set and training labels [50]. By learning the relationship of inputs and desired outputs during the course of many training episodes, the NN is able to predict the output for a specific set of inputs [4] [13]. The goal of this method is to approximate a function that describes the output based on the input and thus correctly label an unlabeled set of input-data [50]. The task of labeling the unlabeled data is called classification [8]. In some instances, this method is also known as Classification-Learning [15]. The training-data can be obtained manually or be created and stored by runs of simulations where the data can be extracted from [60]. Every data element used for training, consists of an input and the true value of the output. The task is to approximate the output as close as possible to the true value by choosing the right parameters or improving the chosen one [61]. For example, in quality-control, workers can collect the necessary data-set for automating the task of finding production faults. In this exemplary data set, the information would consist out of a picture from the part that has to be checked for errors or of a section of that part and the number of actually measured true errors at the part. The algorithm's task is to ensure the quality of the presented part by analyzing it for manufacturing errors. The most prevalent example of Supervised Learning is image recognition. For example, the classification of handwritten numbers [62] [27]. In training, the input is given to the NN, in this case, also called a classifier, to predict the output. If the output value that predicts the label of the input is different from the true value, the learning algorithm responsible for the training process, adjusts the parameters of the NN to improve its performance based on the feedback. In Supervised Learning, the parameters are called weights and biases [47]. The success of the classifier is improved by minimizing the mismatch between the true and predicted output. This is done by calculating the gradient of the mapping-function with feedback from the supervisor. The parameters inside the Supervised Learning algorithm are adjusted by small increments in the direction of the negative gradient. This procedure is also known as "steepest descent". In ML, the updating of parameters, based on the calculated error, is called backpropagation [63]. Data that is used for training and validation of the NN's success, is divided into a training-set and test-set. The test-set is used for the evaluation of the performance of the classifier [27]. The training of the NN is successful if all outputs from the test-set are predicted correctly. In this method, a large database is needed for sufficient training. For real-world applications, this is the underlying problem of Supervised Learning. In the case of complex problems, the building process of a large enough database, is turning out to be unfeasible [64]. In the case of the scheduling of production-orders, Supervised Learning can be used as a tool to predict the problem of when or where a machine-failure might occur. This prediction can be used to adjust the schedule accordingly before the beginning of production [27]. However, an in-time rescheduling of all production orders in the case of machine-failure is not possible with

Supervised Learning. The basis of this problem lies in the data that is needed for the training of the classifier. For every possible machine failure which affects the process times, a new production schedule must be created. The content of such data-set would contain the current state of production as the input and the optimal prioritization of products and process steps as labels. This is not possible due to the sheer amount of options. It is shown that it is not possible to improve production-control with Supervised Learning [8]. The application outside of the area of manufacturing are for example cancer research, credit rating or food quality control [27]. The crucial deficiency of Supervised Learning is that the learning process is very static and can not be incorporated into the dynamic environment of a manufacturing area where frequent rescheduling is required [46].

Figure 2.3 [65] gives an example of how the task of damage detection can be automated with Supervised Learning. The algorithm is capable of autonomously finding the area of damage on the car. The same principle can be applied to production processes in the form of quality control.

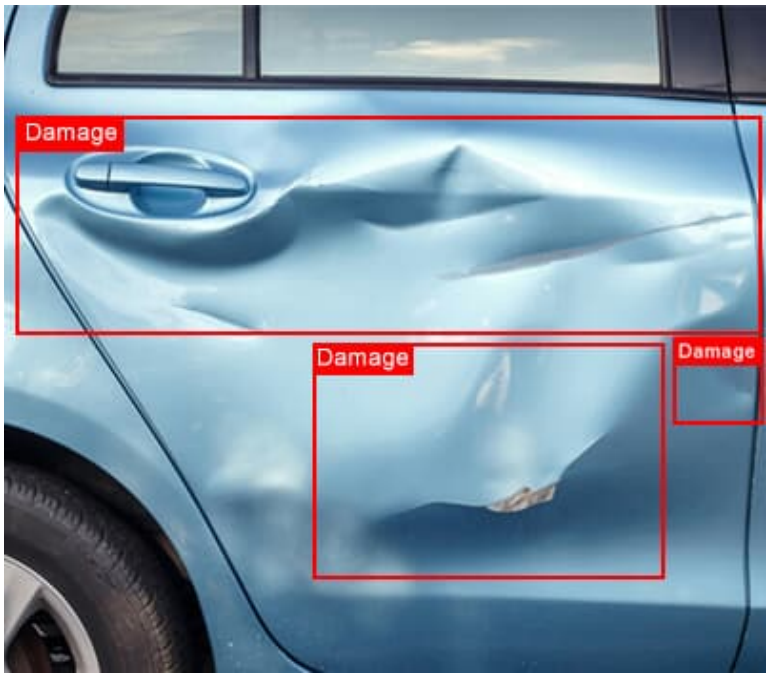


Figure 2.3: Example of damage detection with Supervised Learning [65]

2.4.3 Function and Application of Unsupervised Learning

Unsupervised Learning is very similar to Supervised Learning (see Chapter 2.4.2). Although in this case, the training data is not labeled. The main objective is grouping or clustering of the presented data based on their hidden or obvious features [4]. The biggest difference in comparison to Supervised Learning is, that no feedback is given on the clustering. The data is grouped in clusters by self-extracted features from all input elements of the data-set [27]. This is helpful if no labeled database is available, too costly or time-consuming to create [66]. The range of application of Unsupervised Learning for scheduling in production-control-systems is limited as no new information is generated that can be utilized in scheduling [15].

2.4.4 Function of Reinforcement-Learning-Algorithms

This chapter covers key components of Reinforcement Learning in the context of AI. The necessary elements are described, and the interaction of these elements is explained.

RL is the third subcategory of ML (see Chapter 2.3). It has many fields of application and can meet all the requirements for in-time production scheduling [56]. It is not feasible to replace specialized scheduling-systems with RL-algorithms. RL relies on a recurring interaction with the environment and is not designed to solve static problems [67] [46]. RL has two major components: an environment and an agent [54]. The main difference to other ML-methods that use labeled data is that in RL, no explicit answer is given on how a problem is expected to be solved. Instead of that, the actions of the agent in the environment are rewarded with the help of a specific reward-function and the cumulative reward is attempted to be maximized [46]. The learning process in RL is similar to the way of learning of humans or animals [68]. The advantage of RL is that no supervision or database like in Supervised Learning is needed for training of the NN (see Chapter 2.4.2). RL is the only method that solves the issue of achieving a long-term reward by interaction without human supervision or a database in a dynamic environment [46].

The **action-space** in RL includes all the possible actions that can be chosen by the agent to be executed in the environment. The agent can select only one of those predefined actions. Most of the time, this number of possible actions is finite. In a finite action-space, it is possible for the agent to select each action at least once if enough selections are performed randomly for long enough. For example, the action-space for finding the exit in a labyrinth is defined

as going left, right, forward or backward. These types of action-spaces are called discrete action-spaces [69].

A special variation of the action space is a **continuous-action** space. In this case, the agent returns a numerical value that can be interpreted and executed by the environment. The possible values are not explicitly defined. The only limitation is the range of the values from the minimum value to the maximum value. All possible values and their correlating actions can not be presented as a list as they are infinite. This means that the agent can not explore every single action and its effect or even memorize it. An example of a continuous action-space is the torque of an electrical motor. The torque is not defined as a constant or step function. For example, it can be set to unlimited torque settings like 10Nm, 9.99999Nm, 9.99998Nm et cetera. It is impossible for the agent to learn every outcome of a specific torque setting by trying them out, as the options are simply infinite. [70] [71] [46]

A **reward** is a numerical value, without a unit, returned to the agent, calculated by the reward function that is indicating how helpful the performance of the agent is to solve a problem [46].

The **agent** is the element that can interact with the space it is currently in, also called the environment, multiple times to solve a specific problem [64]. This interaction can be called "trial and error learning" [46]. It makes it possible to expand the knowledge of an agent by trying new actions. By observing the environment, the agent receives information, also called the state, and chooses from a set of predefined actions that can influence the environment. By repeating this cycle multiple times, the agent tries to solve the presented problem [22]. The main objective is choosing a suitable action with the most expected reward for every situation the agent finds itself in [72]. In the very first interaction with the environment, the untrained agent does not know anything about how it is affected by certain actions or what rewards to expect [23]. This action is comparable to an infant that is learning to walk. It doesn't know anything about gravity or the forces acting on its body, but over time it learns to understand how certain actions lead to different results. In this case, falling is considered as a negative reward and thus should be avoided and steady walking is regarded as positive. The main goal is to maximize the cumulative sum of rewards. This defining method of RL can be used in scheduling of production orders [22] [4]. Like a momentarily high production rate, short-term rewards are ignored in the pursuit of a higher long-term reward, which is an overall high production rate. How much the long-term reward is preferred over the short-term reward is defined by the constant gamma (γ). If gamma is close to 1, only long-term rewards are pursued. If gamma is close to 0, only short-term rewards are pursued [46].

In more detail, the agent will choose actions to transfer in a more "valuable" state to gather more rewards in the future. The value of a state is the sum of all expected rewards if starting or continuing from that state. Figure 2.4 [46] outlines the interaction of an agent and its environment. At first, the agent observes the environment it is currently in. The information that is included in the observation is called the state (S in Figure 2.4). The agent chooses an action (A in Figure 2.4) to influence the environment. By influencing the environment, the agent transitions into a new state. For transitioning into the new state, a reward (R in Figure 2.4) is given by the reward-function that is part of the environment. This interaction is repeated again if the problem is not solved. [46]

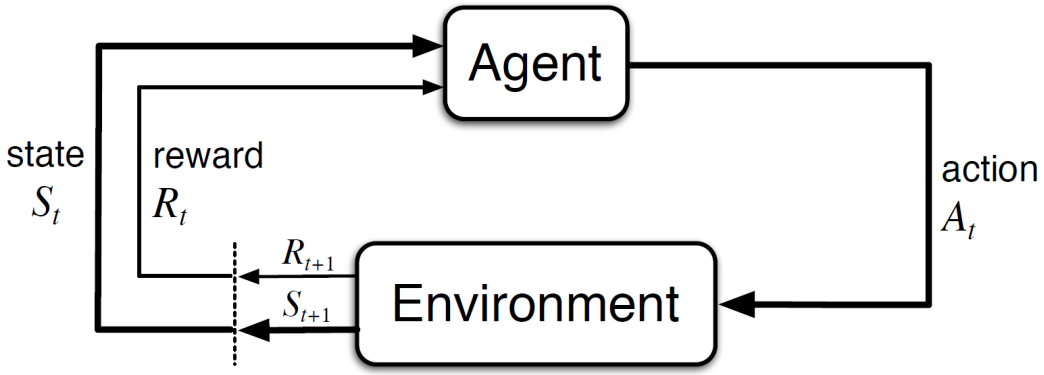


Figure 2.4: The agent–environment interaction [46]

The received rewards are summed and saved to calculate the gradient to improve the NN. The learning process in RL is similar to Supervised Learning (see Chapter 2.4.2). The expected values of visited states are compared to the actual values of the states. With the help of the gradient, the parameter of the NN are adjusted to improve the prediction of the expected rewards and values. It is important to point out that the agent itself has no record of past situations or actions. In this case, the agent does not differentiate what actions are chosen in the past or how the agent ended up in the current situation [54] [46]. For the learning process, a dedicated buffer needs to be implemented that stores the states and actions as well as the rewards [73].

The implementation of only one agent interacting with one environment is called a **Single-Agent-Setup**. In this case, the agent is referred to as a **Single-Agent** [74].

A **multi-agent** approach is where multiple agents interact with each other in one environment. This principle is applied in RL if the problem is not solvable by only one agent [75]. The most prevalent example is a video game where two players play together and help each other to win the game. The environment, in this case, is the video game and both human players are replaced by the agents. Both agents can perceive different information from the environment

and perform individual actions. It is possible to implement the agents in a competitive environment like the video game ”Pong”. In this case, each agent is trying to win the game on its own. [74]

The main difference from a multi-agent setup to a single-agent setup is that each learning agent in the multi-agent setup must explicitly consider the other agents in the same environment. This can cause a problem as all agents are changing their behavior over time. Their actions have to be coordinated with each other for successful problem-solving. Further, the rewards have to be distributed in such a way that every agent knows exactly how to improve its performance in the future. [75]

From now on, the singe-agent in a single-agent setup is called just agent.

Figure 2.4 shows that the **environment** is the only source of information for the agent. It contains information that the agent receives and information it does not receive as well as the reward-function [46] [29]. The agent needs the information to be presented in a certain manner. Normally this is done in the form of an n-dimensional array, which is called a state-vector [67]. An exemplary representation of the interaction of the agent and the environment is a vacuum-robot that has the task of cleaning a room. Figure 2.5 [76] gives a visual example of how a vacuum-robot observes its environment. In this case, the environment is the room, including furniture, plus all physical elements of the robot, like the battery and dust compartment. The agent that is the software of the robot receives all information from the sensors and decides what to do in the next steps [46].

Figure 2.5: Vacuum-Robot observing its Environment [76]

Most of the time in the application of RL, the environment is simulated to increase the speed of learning and exclude the task of digitalizing analog signals. One of the advantages of a simulation in comparison to a real-world environment is that inputs and possible outputs can

be adjusted easily and training can be separated from execution [26] [13]. This is one of the reasons why Cyber-Physical-Production-Systems are gaining in popularity. As soon as the agent reaches a certain success-rate by interacting with a simulation, it can be transferred in the real world without spending any more time on training [67]. The interaction of the agent with the environment is subdivided into cycles (see Figure 2.4). In each cycle, the agent decides on what to do, to receive the highest possible long-term reward. One completed cycle is from now on called a step. If the defined goal or the maximum number of iterations is reached, the simulation of the environment is resetted. The number of cycles from the first interaction to the last is called an episode.

The agent chooses actions based on the information that is received from the environment [46]. Therefore it is important to consider what information is given to the agent. The agent has no memory of past states, so it is necessary to present the state as a **Markov-state** [36] [77]. A state has Markov-property if all relevant information is contained inside the state that makes it possible to fully reconstruct the environment without knowing the past states and actions [13]. Regarding the example of the vacuum robot (see Figure 2.5), a Markov-state would include the position of the robot, the battery status, already cleaned spots and the amount of dirt in the dust compartment. Knowing all this information makes it possible to reconstruct the environment without knowing what sequence of action has led to this state. By using more variables than necessary to describe a state, the agent runs into the problem of understanding which actions influence what values in the state-vector [60]. It is absolutely necessary to express the RL-problem as a **Markov-Decision-Process** (MDP) by encoding all necessary information in the state-vector [46]. This is one of the challenging tasks for the implementation of RL in the area of production-control because not every necessary information is easily accessible [54]. If not all information necessary from the environment is available, it is possible to use specifically designed algorithms that can solve partially observable MDPs to mitigate this problem [78] [79]. These algorithms are not further discussed in this thesis.

The **reward function** in RL is a crucial element for objective-driven problem-solving [22] [64]. It is a mathematical function that rates the agents' actions. The correct implementation of a reward function as a tool in RL that actually serves the desired purpose can turn out to be very difficult [50]. It serves a guiding purpose by observing and rating the agent on its performance [7]. Every action by the agent is rated by the reward function to indicate how helpful it is to achieve the desired goal [22]. The agent does not know how the rewards are calculated because the function is part of the unobservable part of the environment [80]. The correct definition of the reward function is mandatory for successful problem-solving [10]. All expected future rewards from a state are summed up to the expected value of that state. All the partial rewards received for executed actions are defined by the reward function. By

considering the value of a state, it gives the agent the possibility to neglect short term rewards for a globally higher long-term reward [4]. This is the main process of working towards a high global reward [13]. Figure 2.6 gives a visual representation of how the agent has to consider all following actions to solve a problem as efficiently as possible. In this example, the agent can choose two ways to reach the desired destination. By choosing the longer bus ride in the beginning, the agent will reach the destination faster than by selecting the shorter bus ride as it is coupled with two ten minute walks.

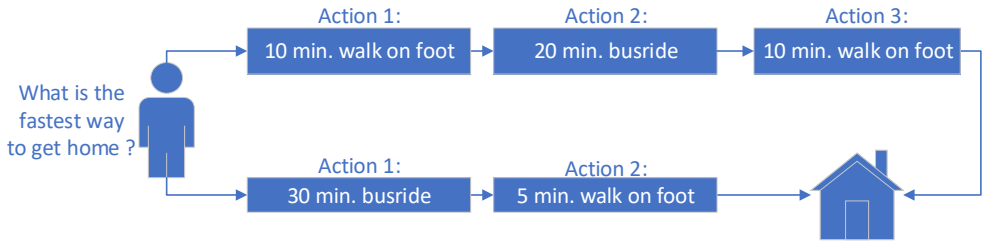


Figure 2.6: The agent has to consider all following actions to solve the problem efficiently

The reward distribution for positive actions, that bring the agent closer to its goal, should be modeled exponentially [46]. This means that improvement is always rewarded more than deterioration is punished. High rewards received by the agent should indicate sufficient performance [67].

The biggest disadvantage of ML is the necessity of retraining in the case of slight changes. The origin of this problem is that the agent learns to optimize its actions based on the defined reward function. If the function or environment is changed, the knowledge of the agent turns unusable for the new purpose [58]. RL-problems that have a changing goal over time are called ”non-stationary problems”. Newly developed methods propose solving such non-stationary problems by disregarding the accumulated knowledge from old iterations and focusing more on recent rewards [46].

On a biological level, the reward-function represents pain and pleasure. When a human being is learning to walk, the pain of falling is a signal stating that the actions leading up to the fall, did not help achieve the desired goal of walking. In the case of the vacuum robot, the trade-off between long-term and short-term rewards is easily explainable. If the main goal of the agent is the cleaning of the designated area as quickly as possible, it has to decide on the optimal time on when to recharge and empty the dust bin. The continuous cleaning with low battery power, far away from the charging station, can result in stranding. The nonstop

cleaning might bring many short-term rewards in the beginning, but the overall global reward is diminished by the punishment of the drained battery and the inability to move. [46]

A **discrete action-space** is a finite amount of actions that the agent can perform. The effects of all actions can be learned over time. This is not possible for a **continuous action-space**. This problem is prevalent in all applications where a continuous action-space is necessary [81]. A clear example of a problem with a continuous action-space is a robot that has the task to walk a certain distance on two legs. It has four motors in total, placed on its knees and hip joints. The motors have a maximum torque and can be applied forward and backward. How much torque in between the set range is applied by each motor, is controlled by the agent [82]. It is impossible to explore every single combination of torque in every possible state to form an optimal prediction for maximum rewards [81] [46].

In discrete action-space problems, the agent has the possibility to choose one action from the available set of actions. For example, the agent chooses one action X from the set of possible actions: A,B,C,D
In a continuous-action space the agent has the task of choosing one action X or multiple actions from the infinite set of: aaa, aab, aac, aad, aae, aaf, aag, aah, aai,...

In problems where an explicit value like, for example, 5.34 or multiple of them is expected as the output, it is beneficial to implement an **Actor-Critic** approach [83].
All details of this method cannot be explained to the full extend in this thesis. The Actor-Critic approach is part of "Temporal-difference" methods [84]. This method is very helpful if an agent has to decide on one action from an infinite amount of actions. An infinite action-space is not fully explorable [84]. Instead of calculating the best single action, based on the current state, a new method is implemented that can work with these continuous action-spaces. Even though this approach introduces more complexity, it is the only way to reduce the required computational time to select an action from an infinitely large action-space [46].

As mentioned in the name Actor-Critic, this approach has two elements: the actor and the critic. The agent and the critic both have a NN. The agent is responsible for selecting an action that will result in the highest possible reward. This is different from the previously described process where the agent is selecting the action based on the value of a state it will transition to. The role of the critic is to approximate the value of possible actions. [46]

In an Actor-Critic approach, the actor is from now on, also referred to as the agent. For further information on this topic it is referred to [85] [84] [86] [87] [88].

The **buffer** is a storage-element that is responsible for saving all information that is necessary for optimizing the NN and thus improving performance. One unit in the buffer is called a **transition**. This transition includes the state that is fed into the NN, the output from the NN, the state of the environment after the action has been executed and the received reward for transitioning into the new state. These transitions are continuously added to the buffer while the agent is interacting with the environment [89]. By having a buffer, the agent can learn from past experiences and does not only rely on just the current interaction. This can speed up the process of learning and eliminate time-related correspondences [73].

For the actual learning process, it is necessary to define two further variables. The first one is the **learning rate**. It defines by what increment the predicted value of a state is changed [46]. Similar to the mentioned update-process in Chapter 2.4.2, the received reward is spread over the visited states and compared to the predicted value. The values are used to calculate the direction in which the prediction has to be adjusted. The learning rate defines how much the prediction has to change. Figure 2.7 depicts how a high learning rate influences the agents' ability to find the minima. By setting it too high, the actual minima are overshot and can result in divergence. By setting it too low, it requires many more iterations for finding the minima. For optimal results, the learning rate can be adjusted in the learning process, to reduce the possibility of overshooting the target. [90] [91] [46]

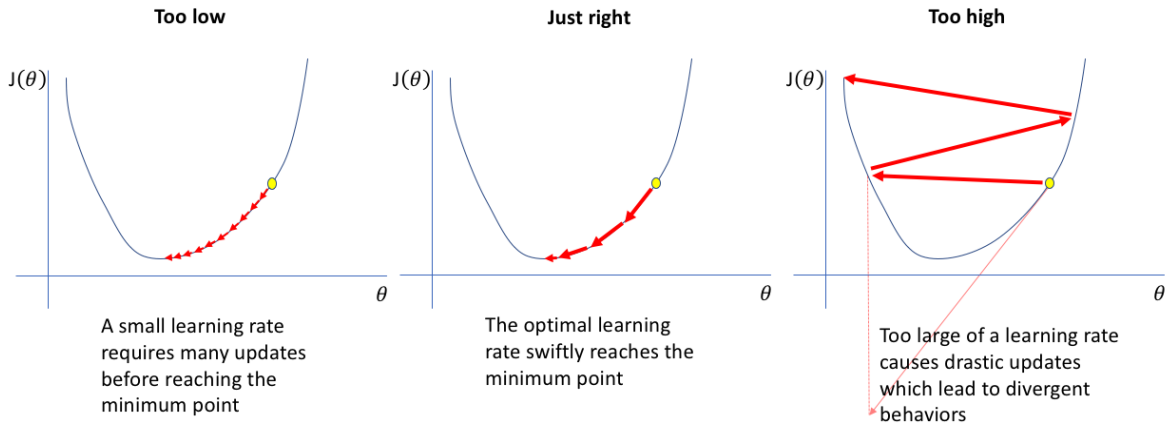


Figure 2.7: Effects of high and low learning rate [92]

The last parameter is the number of samples taken from the buffer to improve the NN. This is called the **batch-size**. This means, that for example 100 samples are taken from the buffer to calculate the gradient to optimize the predictions. By sampling too many episodes from the buffer, the NN loses its ability to generalize in unvisited situations. This is comparable to the agent memorizing all states it has been in. [93] [94] [95]

2.5 Overview of AI in Production Control

For a detailed analysis of the current state of the usage of AI in production control, multiple publications are reviewed based on their suitability to solve the previously described problems (see Chapter 1.2).

2.5.1 Analysis-Process for the Research-Topic

This analysis is structured by imposed requirements (see Chapter 1.4) that will allow a transfer of the chosen AI-approach to the physical, real-world production. Several papers are disregarding crucial factors that have to be considered to enable such an application of AI in manufacturing in the future. A few papers are close to emulating a real-world production in their approaches and only lack minor aspects.

To manage a successful transfer of RL into a cell-based production environment, the requirements are grouped into three categories. The first category are requirements regarding the function and setup of the chosen ML-method, the second category requirements regarding the cells and the third category regarding the aspects of the production. These three over-categories are broken down into multiple subcategories and allow a detailed view of the issue. Every analysis of each over-category will follow the same pattern. The subcategories are arranged in descending order by the number of papers that have dealt with the imposed requirements. At the end of this chapter, a table (Table 2.1) is presented that summarizes the state of the usage of AI in production control and gives a visual representation of the research area up to now. By completing this thesis, the attempt is made to close the presented research gap. This analysis is focused on the approaches and results published in the publications: [10], [22], [60], [13], [36], [77], [67], [25] and [56].

2.5.2 Composition of the chosen ML-method

The first grouping is the composition of the ML setup. It is discussed in three different aspects.
First, the chosen algorithm needs to interact with the environment without any time delays and instantaneously generate actions that are executed in the simulation (R3, see Table 1.1). The online-reaction capability is required not to lose time on repeated calculations of adapted schedules. Each approach of the analyzed papers has the potential to give a quick response to a received input. This is one of the reasons why ML is chosen in every instance. Without

a quick response time, ML would be unfeasible for Online-Scheduling (see Chapter 2.1) and could easily be exchanged for a traditional scheduling algorithm [13].

Further, the focus lies on RL for continuous interaction with the environment. This approach is necessary to be able to continuously adapt to every changing situation. Although RL is the most researched ML-method to control a production, [60] uses a Supervised Learning method to optimize the production rate. The presented idea is to divide the production time into sections. For each time interval, one of the preselected scheduling rules is chosen by the classifier (see Chapter 2.4.2), based on the current state of production. This method eliminates the necessity for a large database on which the algorithm has to be trained but sacrifices performance as no continuous rescheduling is possible. Like in image classification (see Chapter 2.4.2), the NN can spot specific features that correlate to the optimal scheduling rule for this situation. The shorter the time frames for rescheduling are chosen, the better the result of the scheduling rule [60]. All other listed papers are using a conventional RL approach (R1, see Table 1.1).

The third subsection is the number of ML-instances that are implemented in the control loop. A single-agent approach (R2, see Table 1.1) does not require inter-communication of the agents and is therefore easier to implement. In the Supervised Learning approach of [60], it is evident that only one classifier is needed to oversee the manufacturing process and decide what heuristic all products must follow. In RL exists the possibility to implement multiple agents that can act independently (see Chapter 2.4.4). This is called a multi-agent approach and can be implemented in two ways.
[13], [77] and [25] are implementing and using a multi-agent approach. [13] is using one agent for every workstation in the manufacturing area. The goal in this constellation is similar to [60]. Every workstation can choose from preselected heuristics to maximize the production flow. The reasoning behind this is that in that approach, it is not possible to assign different dispatching strategies to different machines by one central agent. Further, it is stated that assigning separate agents to different machines allows for an easy up and downscaling of the manufacturing process. In addition, it is possible to pause the learning process for chosen agents and thus to stabilize the learning of other agents [13].
[77] is implementing a multi-agent system that chooses not the workstations as agents but each planned and scheduled product. Each agent has the option to select the next machine from the routing that will perform one of the required process steps. This procedure is motivated by the idea to reduce the large action-space of a single agent. In this case, the action-space is the number of optional machines from the routing the agent can choose from, which can perform the next process steps.
[25] is using a global agent as well as multiple individual agents. In this method, the global agent is responsible for creating a schedule that incorporates possible machine failures from

the beginning. The global agent replaces the APS-system (see Chapter 2.1). If sudden disturbances occur or the centrally created schedule is infeasible, the individual agents take over to continue the manufacturing process. By learning from the decentralized agents, the global agent can improve on its schedules. Despite the hoped advantages, a multi-agent approach has apparent downsides. It introduces additional complexity to the implementation. For global optimization, goal-oriented communication has to be established between the agents. The reward-function has to differentiate what action of which individual agent helps to achieve the goal and decide what simultaneous actions are beneficial. Only a clear subdivision of rewards allows every agent to improve. [25] [77]
In this over-category [10], [22], [36], [67] and [56] took the necessary requirements into consideration.

2.5.3 Requirements regarding the Manufacturing Cells in Manufacturing

The second over-category are the requirements regarding the cells in the manufacturing process. Machine breakdowns (C4, see Table 1.1) are the most important physical incidents in the manufacturing process for production control. These incidents make it difficult to use mathematically perfected scheduling algorithms to calculate the optimal sequence and timing of process steps. Nevertheless, machine breakdowns are a common and unavoidable occurrence in a manufacturing system. The possible failures are known, but the timing is impossible to predict [1].
A lot of research groups that have no ties to physical manufacturing areas, tend to disregard this crucial element. In the chosen list of research papers, [13] and [36] are ignoring this requirement completely. [10] explicitly mentions the machine breakdowns but limits the number of broken machines to only one. This eliminates the need where a global view of the manufacturing process is necessary to choose how to proceed further.
By disregarding machine failures, the implementation of RL loses big parts of its effectiveness as the dynamic of the manufacturing-system is strongly reduced.

A cell-based manufacturing-system (C1, see Table 1.1) is necessary to be able to handle the growing number of custom-made products (see Chapter 1.1). A cell-based manufacturing-system provides the option to switch to alternative machines if the planned one is out of service or the required employees are not available. Without a cell-based manufacturing-system, it is impossible to choose alternative machines for certain process steps, to reduce the effect of disturbances. This is one of the biggest problems in conveyor-belt style manufacturing [96].
In this case [22], [67] and [56] are not focused on a cell-based manufacturing-system. [56] is focused on wafer-fabrication in a linear looped layout. If the action of the central agent is required, the agent can choose what products from the machine-buffer of a machine should

be prioritized. [22] is considering a cell-based layout and ignores the sequential nature of the manufacturing process. Products are manufactured on one machine and do not require further process steps. [67] does not mention any design and is primarily focused on optimal order dispatching in the semiconductor industry.

In a real-world manufacturing area, machines can be used for multiple purposes (C2, see Table 1.1). As an example: A milling machine can be utilized for its main milling purpose but also replace a drill press. The milling machine can be a viable option if the drill press is not operational for a longer period of time. [10], [60] and [77] took this factor into account in their publications. It's important to point out that by implementing this function out of the requirements in the wrong way, it can lead to a noticeable deviation from the real world. In the case of [77], it is assumed that each machine can perform every process step. This shifts the problem to a combinatorial problem and opens the option to use more alternative machines than feasible in a real-world manufacturing environment and therefore, should be avoided.

Originating from the multiple-skill-requirement is the efficiency requirement of the possible skills (C3, see table 1.1). The ideal image being that multi-skilled machines are capable of performing each task with the same efficiency as other specialized machines in the manufacturing process, is an unattainable vision. Regarding the example brought up earlier, it is not possible for the milling machine to be as efficient as the drill press because of the necessary setup and machine programming times of the milling machine. Likewise, an often required process step with high precision requirements takes more time on the drill press than in the milling machine. This aspect is passed over by most of the current studies [10] [22] [13] [36]. Only [60] fully fulfill all requirements in this section. In some papers, it is not clearly evident if those efficiencies are implemented in the required setup times for different products.

2.5.4 Requirements for Manufacturing and Products

Heuristics are not what is desired in the presented use-case (C5, see Table 1.1) (see Chapter 1.3). Heuristics dictate a rule that every product or machine has to follow. This limits the ability of the products to pursue an individually chosen sequence of process steps. Although a chosen heuristic can be optimal for the majority of products, it sacrifices a possible performance boost by limiting the action-range of products that suffer under the chosen heuristic. [22], [36], [77] and [25] are all implementing a simulation where the products move independently. Most of the methods in the published papers following a multi-agent approach, are implementing greater freedom of movement of products than the approaches

using a single agent. Like [13] stated, this is to bypass the problem of creating an agent with a large action-space (see Chapter 2.4.4).

Like the aspect of machine failures, transportation time (C6, see Table 1.1) is something that is a pronounced element in the manufacturing process. It is explicitly implemented in ERP-systems and should therefore be considered in the implementation in the RL approach. This is the aspect that is left out the most. Only [77] is considering the necessary time it takes one product to be transported from one manufacturing-cell to the next. The consideration of time is important because it can be a decisive factor on which alternative machine to choose if the targeted machine is nonoperational.

2.5.5 Research Gap

In summary, we can see a research gap in the implementation of a single-agent RL approach that is capable of online-reactions in a cell-based manufacturing environment. The agent has to consider the machine-skills, their efficiencies and potential breakdowns. In addition, attention has to be given to the transportation time between the machines and the independent movement of the individual products. Table 2.1 gives a visual representation of the required aspects of the analyzed papers. A "*" in the dedicated column is indicating that this paper fulfilled this requirement.

Table 2.1: Related work in the field of online-scheduling with RL in a cell-based manufacturing environment

	Reinforcement Learning (R1)	Single-Agent Setup (R2)	Online-Action (R3)	Cell-Based manufacturing (C1)	Cells with multiple skills (C2)	Skills with different efficiencies (C3)	Machine-Breakdowns are considered (C4)	Products move independently (C5)	Transportation-Time is considered (C6)
[10]	*	*	*	*	*		*		
[22]	*	*	*				*	*	
[60]		*	*	*	*	*	*		
[13]	*		*	*					
[36]	*	*	*	*				*	
[77]	*		*	*	*		*	*	*
[67]	*	*	*				*		
[25]	*		*	*			*	*	
[56]	*	*	*			*	*		

3 Methodology

In this chapter, the necessary requirements are laid out for implementing an RL-based production control system and a general systematic approach is presented. This method can be applied to control machines or products in a cell-based production environment.

3.1 Use-Case and Overview of the developed Method

Figure 3.1 gives a visual representation of how the following method interacts with the manufacturing process. Multiple machines perform various process steps on products. As soon as an error in the manufacturing process occurs that influences the production, the agent provides a new schedule, including optimal prioritization of process steps and products. For calculating the optimal decision, the agent is provided with information on the current state of production and instantaneously returns the result. From this point on, the agent continuously receives information from the manufacturing process and controls it until all products are finished.

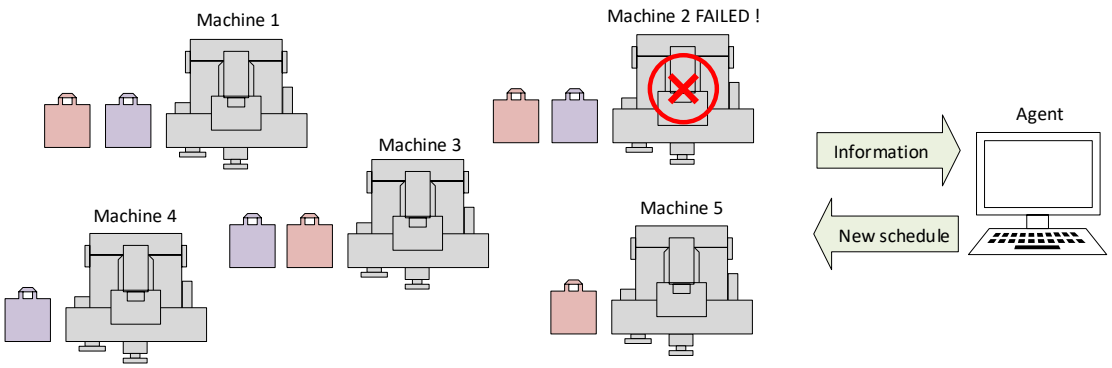


Figure 3.1: Use-Case of Online-Scheduling in the Manufacturing Process

The following developed method is divided into seven distinct steps (see Figure 3.2). The following chapters explain each of the required steps in detail.

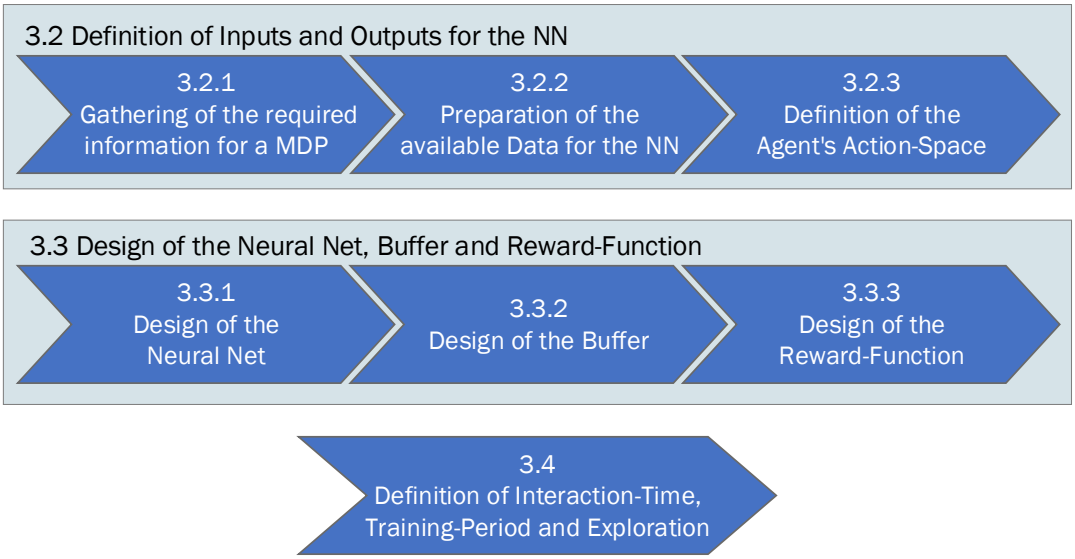


Figure 3.2: Required steps for the Implementation of an RL-based Production-Control-System

3.2 Definition of Inputs and Outputs for the NN

In the following, it is explained how data from the manufacturing process has to be prepared in order to serve as an information source for the agent.

3.2.1 Gathering of the required information for an MDP

The first step is the gathering of all information from the manufacturing simulation. The environment is the only source of information for the agent (see Chapter 2.4.4). In the presented use-case, a simulation is needed that the agent can interact with and information can be extracted from. This simulation can be a simple mathematical model or a complete digital twin. It is possible to extract data from the physical, real-world production by high sensorization or transforming analog signals to digital. The manual conversion of analog signals, like information from the written form, is very time-consuming and thus limits the online-action capability of the agent. In the following, it is assumed that a simulation is available and no conversions are necessary.

The most important part is the amount of information that can be extracted from the simulation. It is required to state the problem as an MDP (see Chapter 2.4.2). It is possible to combine multiple simulations and use additional information sources that help to achieve the Markov property. The result of combining multiple simulations is a new environment. It is always viewed as a single complete unit. If the information from the simulation and other sources is not enough to create a Markov-state, it is not possible to implement the presented RL-approach. The Markov-state for a simple manufacturing process can contain the following information:

- The number of products with their actual positions
- What processing steps are already completed and which are still necessary on each product
- The number of machines and the required working times for every skill
- The routing options for every product
- Occupancy status of every machine
- In case of machine failure, how long it will take the machine to be back operational

For a complete MPD, more information like remaining transportation time is needed. The definition of a state is individual for every production system and has to be created accordingly. The more complicated the design of a cell-based manufacturing system is, the more information has to be contained in the state. The factory should be fully reconstructable from the information that is given to the agent without being dependent on further information from past events (see Chapter 2.4.4).

For a complete reconstruction of a simulated manufacturing process without prior knowledge, a set of constants needs to be included in the MDP. For example, the time it takes to transport a product from one machine to all other machines. This information is not available if only the remaining transportation time is included in the Markov-state. In the case where an agent is being trained on only one simulated factory, it is possible to leave this information out. This refers only to the constants that are not changing during the course of one episode and in between runs. By interacting with the environment, the agent is expected to figure out what information is implied and how to work with it. The agent will learn and memorize the transportation time by performing random actions and observing the changes in the environment. This knowledge will be stored in the NN. [46]

3.2.2 Preparation of the available Data for the NN

The information from the manufacturing process can be present as scalars or ,in some cases, like the transportation-time between machines, as matrices. The NN requires the information

to be organized and presented as a single array (see Chapter 2.4.4). Matrices have to be flattened to an array and combined with single scalars and other arrays. The order of specific elements in the compiled array is not specified. The only requirement is that the order stays the same over the course of every training episode. Before feeding the information to the NN, it is necessary to scale the numbers to a uniform range. Normally this is done in the range from -1 to +1 or from 0 to +1 [60]. There are two ways of doing this scaling. The first one is to combine all numbers into a single array and then scale the numbers to the desired range. The second way is scaling before the compilation into an array. The second way is preferable if low and high numbers are present in the unscaled state-vector. The pre-scaling of individual elements is important to preserve the features of the production. The following example shows how to prevent loss of precision by scaling parts of information before the combination into an array. The presented matrix and the two arrays in (3.1) can represent any kind of information like temperatures or occupancy status.

$$\begin{bmatrix} 1 & 3 & 4 \\ 6 & 6 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 100 & 300 & 200 \end{bmatrix} \tag{3.1}$$

At first, all elements from (3.1) are placed in one array without prescaling. The result is presented in (3.2).

$$\begin{bmatrix} 1 & 3 & 4 & 6 & 6 & 1 & 1 & 1 & 0 & 100 & 300 & 200 \end{bmatrix} \tag{3.2}$$

Then the array is scaled to the rage from -1 to 1. The highest number is assigned 1 and the lowest -1. The remaining numbers are adjusted accordingly. This method of scaling is called Min-Max-scaling. The result is shown in (3.3).

$$\begin{bmatrix} -0.99 & -0.98 & -0.97 & -0.96 & -0.96 & -0.99 & -0.99 & -0.99 & -1 & -0.3 & 1 & 0.3 \end{bmatrix} \tag{3.3}$$

All information that is encoded in the matrix and the first array is strongly being suppressed by the high values from the third array. As the NN is modeled after the function of biological neurons, the NN does not drastically differ between numbers that differ by low decimal places. To prevent this, all elements can be scaled beforehand to keep the features. The result of prescaling every single group of elements and combining it to one array is shown in (3.4).

$$\begin{bmatrix} -1 & -0.2 & 0.2 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 0 \end{bmatrix} \tag{3.4}$$

With this method of scaling, it is possible to achieve reasonable results in some cases, though some information is still being lost. This procedure is only valid if a matrix or array contains numbers close to the absolute possible minimum as well as the maximum value. Otherwise,

a lot of sensitive information can be lost. The example in (3.5) shows how, different arrays are scaled to one identical array if the Min-Max-scaler is applied even though the absolute value differences are not the same.

$$\begin{bmatrix} -2 & 2 & 0 \end{bmatrix} \begin{bmatrix} 100 & 300 & 200 \end{bmatrix} \begin{bmatrix} 99 & 101 & 100 \end{bmatrix} \text{ are scaled to: } \begin{bmatrix} -1 & 1 & 0 \end{bmatrix} \tag{3.5}$$

This loss of information can be ignored in some cases but can turn out to be vital in others. For example, if the arrays represent the temperature of machines. The objective of the agent can be to turn off machines if they reach a certain temperature (for example, 100 degrees). The information on the absolute temperature is lost by using the Min-Max-Scaler. The agent only receives information about the relative temperature of the machines and thus is not able to understand the state and solve the problem.

To solve this, a range has to be defined that includes all values that can be reached in the state. In the case of temperature, it ranges from room temperature to the temperature of failure. In the case of downtime, it is set from the minimum to the maximum time-to-repair. This implies that the range is known beforehand. It is important to choose the range as precise as possible to avoid suppressing important information. The following examples in (3.6), (3.7) and (3.8) show how to correctly scale matrices and arrays with a defined range.

$$\begin{bmatrix} 2 & -2 & 0 \end{bmatrix} \quad \text{is scaled from a range [-3,3]:} \quad \begin{bmatrix} 0.66 & -0.66 & 0 \end{bmatrix} \tag{3.6}$$

$$\begin{bmatrix} 100 & 300 & 200 \end{bmatrix} \quad \text{is scaled from a range [0,500]:} \quad \begin{bmatrix} -0.6 & 0.2 & -0.2 \end{bmatrix} \tag{3.7}$$

$$\begin{bmatrix} 99 & 101 & 100 \end{bmatrix} \quad \text{is scaled from a range [50,150]:} \quad \begin{bmatrix} -0.02 & 0.02 & 0 \end{bmatrix} \tag{3.8}$$

For some applications, it is beneficial to specifically increase specific values in the state-vector to amplify the importance of that information. To amplify the importance of special values, they can be increased prior to scaling. The second way of conveying important information to the agent is by placing the value multiple times in the state-vector. Both approaches require extensive testing to achieve the desired performance of the agent. As an example, a state-vector is created from the two elements in (3.9).

$$\begin{bmatrix} 1 & 3 & 4 \\ 6 & 6 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \tag{3.9}$$

The result after combining and scaling to a range from -1 to 1 is shown in (3.10).

$$\begin{bmatrix} -0.666 & 0 & 0.333 & 1 & 1 & -0.666 & -0.666 & -0.666 & -1 \end{bmatrix} \tag{3.10}$$

Manual multiplication of the array with a randomly chosen number like 15 prior to scaling, results in (3.11).

$$\begin{bmatrix} -0.866 & -0.6 & -0.466 & -0.2 & -0.2 & -0.866 & 1 & 1 & -1 \end{bmatrix} \tag{3.11}$$

In the resulting array, the values from the array are clearly dominating and thus are stronger influencing the agent's decision.

In the other method, the array can be appended a second time to the state-vector [67], which results in the array shown in (3.12).

$$\begin{bmatrix} -0.666 & 0 & 0.333 & 1 & 1 & -0.666 & -0.666 & -0.666 & -1 & -0.666 & -0.666 & -1 \end{bmatrix} \tag{3.12}$$

Regardless of the chosen method of scaling or expression of important values, the state-vector has to always stay the same length. This is absolutely necessary as each neuron in the first layer requires an input. The length can not be changed after the NN of the agent is initialized.

3.2.3 Definition of the Agent's Action-Space

The second step is to define how the agent is expected to interact with the environment. To give the agent complete control of all products, he must be able to decide on where and when each of the products moves. In addition to that, he should control what product has to be chosen from the queues to be worked on by the respective machine. By this, the agent has the option to transport a product to a machine and keep it in the queue to wait for the optimal time to start the next process step. If less control of the agent is desired, it can be adjusted accordingly. For example, the election of the next product from the queue for processing can be carried out by the machine itself. If the election process is a constant process on each machine, the agent can figure it out by observing how the environment reacts.

It is possible that the agent issues the command that two products from a queue have to be worked on by a machine. In this case, it is necessary to develop a decision-making algorithm that decides what to do in the case of conflicting outputs. An implicit prioritization can be implemented that chooses to execute the output for a higher value product over a lower value one. The output of the agent is generated by the NN and is only available as a scalar number or set of numbers. Those numbers come from the last layer of the NN. After forward feeding the input through the NN, each neuron on the last layer is returning a number as the output. The number of input-neurons is the number of elements in the state-vector and the number of output neurons is the number of necessary or expected returned values from the NN.

An external function is needed to interpret the output from the NN. This function receives a scalar or an array and returns one action or a set of actions that can be executed in the mathematical model or the digital twin. In the case of Atari-games like Pacman, the output has to be translated into one of the possible actions the game can accept. The actions are the directions, the user or the agent can guide the avatar. There are four possible actions (up, down, right, left) and thus four output neurons [97]. In this case, the NN returns an array with four elements. Each index of the array is associated with one action. The interpretation function finds the index of the maximum element from the output array. The index states what action has to be executed. The same principle applies to the control of one single product in the manufacturing process. Each neuron on the output layer is associated with a possible action. For example, the first neuron is responsible for sending the product to the first machine. The neuron with the highest output value, in other words, with the highest activation, defines what action is executed. In the case of heuristics, the neuron with the highest activation is dictating what scheduling-rule is chosen for the manufacturing process.

When controlling multiple products with a single agent, it is impossible to correlate only one index of the output array to the action for every product. The agent needs the ability to send each product to a different or the same machine as well as keep it in place or take one from a queue to start a process step. The equation to calculate the number of possible combination that determines what every product can do is shown in (3.13). In this equation, n is the number of products and m the number of machines.

$$Combinations = (m+2)^n \tag{3.13}$$

This approach is unfeasible, as for a manufacturing process with ten machines and ten products, 61917364224 output neurons are required. This is not scalable and impossible to train. In addition to that, most of the available output-neurons are not useful and never used. This applies, for example, to all neurons that send all products to the same machine. This scenario is unfeasible, as every product requires different process steps at different times.

To bypass this issue, more information than just one index needs to be extracted from the output array. For this, the output array is divided into sections. Every section is belonging to one product. From each section, the index of the highest value is extracted. This index represents the action to be executed for that specific product. Equation (3.14) shows how to calculate the number of necessary output neurons when this approach is followed.

$$\text{Amount of Output-Neurons} = (m+2) * n \tag{3.14}$$

This makes it possible to reduce the action-space of an agent controlling ten products to 120 output neurons. This approach can be used as well to control multiple machines that can choose from a preselected set of heuristics. The subsections of the output array do not have to be the same size if some products or machines have fewer options to choose from. The number of output neurons can be freely defined based on the necessary requirements for all the products. The extraction-function that finds the index of the highest value and returns an action must be able to read the signals for every possible input, even if all elements are the same and prioritize on its own without user input.

The exemplary output array in (3.15) is subdivided into three sections and translated into actions for multiple products.

$$\begin{bmatrix} 0.0 & -0.4 & 0.6 & 0.1 & 0.2 & 0.5 & -0.3 & 0.4 & -0.5 \end{bmatrix} \tag{3.15}$$

The newly created sub-arrays are shown in (3.16).

$$\left[\begin{array}{ccc} 0.0 & -0.4 & 0.6 \end{array}\right], \left[\begin{array}{ccc} 0.1 & 0.2 & 0.5 \end{array}\right], \left[\begin{array}{ccc} -0.3 & 0.4 & -0.5 \end{array}\right] \tag{3.16}$$

One section is assigned to one product. From every subsection in (3.16), the index with the highest value is correlated to an action for the respective product (see Table 3.1).

Table 3.1: Correlation of Indices, Sub-Arrays and Actions

	Assigned to product	Highest value at index	Associated action
Sub-array 0	0	2	2
Sub-array 1	1	2	2
Sub-array 2	2	1	1

3.2.4 Actor-Critic Approach in Scheduling

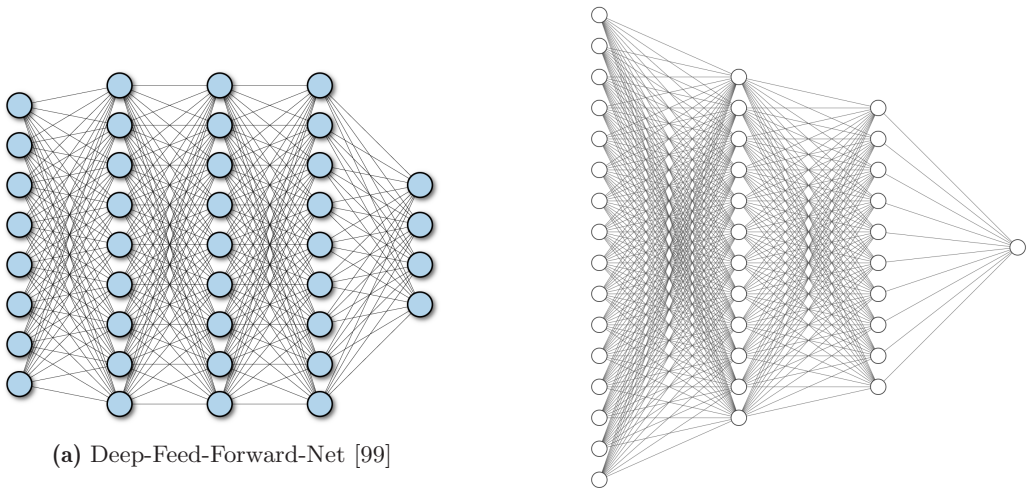
The returned numbers from the NN are not limited to a set of integers. An output-neuron can return a scalar depending on its activation function. From this follows that the output-possibilities of one neuron are infinite. This is not a problem if only one action is extracted from the output array of the NN by finding the index of the highest value and saved by the buffer. This simple principle is used to control Atari-games like ”Snake” with only one input at a time, where no simultaneous inputs are possible. The complexity of this system is significantly increased if multiple values from the NN are considered, and simultaneous actions are possible or required. This raises the problem for the agent of understanding how to combine multiple outputs with infinite possibilities to perform the desired action. This problem is prevalent in all applications where a continuous action-space is present (see Chapter 2.4.4). The same applies to the control of multiple products.
As this problem is not based on a discrete action space, where only one output is expected, it is beneficial to implement an Actor-Critic approach or similar algorithms that can work with a continuous action-space. In the case of controlling all the products with only one output array, the possible values returned from the NN are infinite. In addition, the agent has to learn that different outputs can lead to the same action if the proportions of the elements from the sections are the same.

3.3 Design of the Neural Net, Buffer and Reward-Function

In the following, it is explained how the NN, the buffer and the reward function are designed to achieve the desired goal.

3.3.1 Design of the Neural Net

The necessary number of input and output neurons is explained in Chapter 3.2.2 and 3.2.3. The next step is to decide on how to design the NN in the hidden layers. The hidden layers are all the layers in between the input and output neurons. It is not clear how a NN has to be designed to achieve the best results in the shortest training possible [98]. Multiple designs are proposed for different kinds of problems. Figures 3.3a [99] and 3.3b [100] show two of multiple options for designing a NN.



(a) Deep-Feed-Forward-Net [99]

(b) Feed-Forward-Net [100]

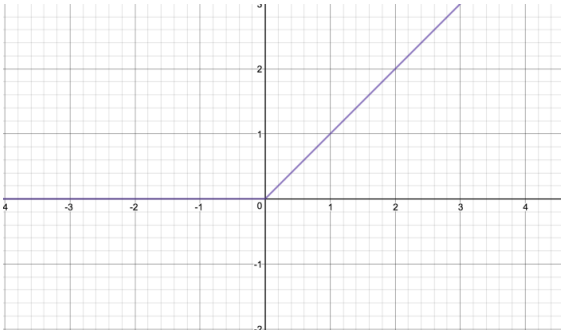
Figure 3.3: NNs with different architecture [99][100]

Figure 3.3a is a Deep-Feed-Forward-Net. The input is fanned out to the three hidden layers and collapsed back to the output neurons. This is a very common design in ML. Figure 3.3b [100] is as well a Forward-Feed-Network but not as deep as the one in figure 3.3a [99]. In this case, the input is sequentially compressed to the size of the output layers. The depth is defined by how many layers are present in the NN. In both of these cases, all neurons of neighboring layers are fully connected. This means that every neuron from one layer is connected to every neuron of the next layer. The chosen connection type allows for more

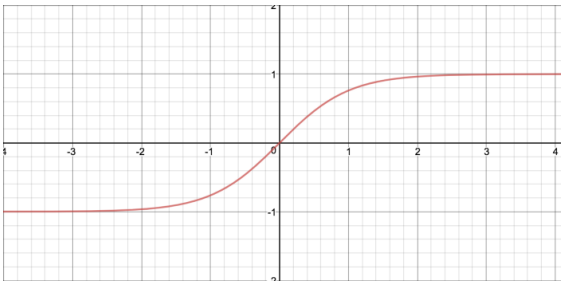
specialized nets [98]. The selection of the best suitable amount of layers and neurons, as well as connection type, requires multiple test runs before finding a suitable architecture.

Each layer requires a definition of the activation function. The activation function defines how the neurons of a layer react to the input. The most commonly used function is the Rectified Linear Unit (ReLU) [51]. There are many more activation functions to choose from and individually defined functions can be implemented as well. Their uses and details can not be covered in the scope of this thesis. For more information on the activation function, it is referred to [101]. To reduce the complexity of the NN design, the activation function can be set to ReLU on every layer and the connection type as fully connected.

Figure 3.4 shows two different activation functions.



(a) Rectified-Linear-Unit-Function [102]



(b) Tangens-hyperbolicus-Function [102]

Figure 3.4: Two different activation functions [102]

The ReLU-functions (see Figure 3.4a) sets all negative inputs to 0 and does not change the positive inputs. The tangens-hyperbolicus function (tanh) contracts all possible inputs to the range form -1 to 1 (see Figure 3.4b).

3.3.2 Design of the Buffer

For the learning process, a buffer (see Chapter 2.4.4) needs to be implemented. The buffer is responsible for saving all information that is necessary for optimizing the NN and thus improving performance. This includes the state that is fed into the NN, the output from the NN, the state of the environment after the action has been executed and the received reward for transitioning into the new state. All states following the first one are saved at first as a state after the execution and in the following iteration as a state before the execution of

the action. The last piece of information is an indicator that defines if the problem is solved or not. In complex problems with large state-vectors, the capacity of the physical storage, usually the RAM, can be exceeded. Therefore it has to de defined how many elements are stored in the buffer and determined if the physical storage capacity is enough. To eliminate the error of memory overflow, the buffer has to overwrite old elements to replace with new ones if the limit is reached.

Equation (3.17) shows how to calculate the necessary storage for 1000000 elements in a problem with 200 inputs and 50 outputs, if all values are stored in foat64.

$$(200 * 2 + 50 + 1 + 1) * 64 \text{ bit } *1000000 \text{ elements} = 3,6 \text{ GB} \tag{3.17}$$

It is important to not set the limiter of the maximum elements stored too low, as a phenomenon called Catastophical Forgetting can occur [103]. In this scenario, the buffer only contains states and actions from the latest successful runs. If the agent finds itself in less ideal situations than usual, the NN can not apply its knowledge to calculate what to do for the best possible outcome, as all recent updates of the NN are based on successful episodes in the past. The agent has no explicit access to the buffer. It can not compare the current state to similar old ones and, on that basis, decide what to do. This lookup-table-method is not scalable and runtime inefficient (see Chapter 2.4).

3.3.3 Design of the Reward-Function

The reward function is the tool that informs the agent of its success and failures. It needs to be correctly defined for every purpose. It is rating the agents' performance on how close the agent came to solving a problem. The reward function receives the current state or parts of it as an input and calculates the reward (see Chapter 2.4.4). The reward can be negative, as well as positive. High positive rewards indicate good performance. In the manufacturing process, for example, the overall goal can be the completion of all orders for the day or a certain period. This goal is not yet sufficiently defined as no metric is given on which the agent can optimize on.

Possible options are the completion of all orders as quickly as possible, with the lowest possible transportation costs, with the lowest possible transportation time or with an equally spread machine utilization or lowest manufacturing cost. Some of these reward functions, like cost reduction, are easily implemented and return the cost spend on every timestep. In other cases, the rewarding or punishment of the agent can only take place if the agent solved the

problem or the time of one episode has run out. This is true for cases like time optimization. The agent can choose multiple times to deliberately wait before starting a process step and consequently sacrifice the current production rate, but achieve a faster overall completion of all products. The reward function calculates based on the current state of products and time, how optimal the state is and returns a reward. The buffer always saves the same set of variables in its memory. If no reward is given for one transition, the buffer will return an error message. Therefore if the agent is rewarded only at the end of a training episode, all prior rewards are set to 0.

It is possible to implement sub-goals in the reward function. By combining multiple reward systems, the agent will try to find the optimal balance between them for maximum reward. It is very important not to directly instruct the agent on how it should behave by distributing rewards for certain actions. For example, if the agent's task is to finish all orders, it's not beneficial to distribute rewards for finishing each process step. These rewards can hinder the agent from achieving its global long-term goal by chasing the short term rewards. Likewise, it is not helpful to instantly punish the agent for calculating and executing non-optimal decisions as the primary goal can shift to avoiding those negative rewards. The agent must be incentivized to improve only by the reward-function and not by a predefined human reward for certain decisions. [46]

3.4 Definition of Interaction-Time, Training-Period and Exploration

The basic concept of RL is learning through interaction. Therefore it is necessary to define how long an agent can interact with the environment in one episode. This time-frame is defined by the number of possible interactions, also known as steps or explicitly stated as a time limit. In the example of a child that learns to walk (see Chapter 2.4.4), a time interval is not clearly defined as this is a continuous task over a couple of years. In video-games like car-racing, the time frame is clearly stated. The finishing line has to be crossed in less time than is given. Otherwise, the player is eliminated and has to start over. However, every interaction does not need to be the maximum possible duration. The race can be finished before the available time elapsed and then started over to improve performance.

For an optimal learning process, it is important that this time-frame is properly selected. For example, in video-games, the duration the player can interact with the game is not limited to the minimum required time to win the game. In this case, it is very difficult for the player to understand what to do for improvement. The agent, like an inexperienced player, needs more time to learn the dynamics of the game. The same principle applies to the agent in

the manufacturing process. By having more time to understand the interconnected relations of the manufacturing process, more learned efficiencies can be exploited in future episodes to receive more rewards. It is equally important not to extend this time-frame for too long. The drawback of too many interactions is that the reward has to be distributed over many past states. By this, the true value of a state takes longer to calculate (see Chapter 2.4.4). Multiple experiments showed that for achieving reasonable results, the interaction time has to be carefully selected [104]. It has to be at least as long as the minimum required time. This implies that the minimum required time can be calculated or is known beforehand to give the agent enough interactions in one episode for learning. This metric can be calculated by production-control-systems like ERP-software in combination with APS-systems.

To make full use of the available training time, the agent can transition to states with less than the highest predicted values to validate or correct the prediction. This process is called exploration. The NN always returns an action that will predictably result in the most expected reward for the current state by default. Those predictions can be incorrect by large margins. It takes many iterations to correct the imprecise predictions. In some cases, the predicted values of a better, yet unvisited state, can be vastly lower than a non-optimal state. If no exploration is implemented, the agent will never decide to visit the predicted lower value states. The predicted value will not be corrected and thus, the success of the agent is limited. [46]

This is prevented by forcing the agent to execute unrequested and seemingly non-optimal actions. The purpose of this is to visit more unvisited states to develop a correct prediction of their values. This is normally implemented in a way where the output of the NN is ignored and all elements of the array are overwritten with randomly generated numbers. This means that during the training, a set percentage of output arrays is ignored and replaced randomly. The new randomly generated output is treated by the interpretation function and buffer like a normal output. Random actions are rewarded the same as those generated by the NN. [105] [106]

Another way of implementing exploration is by adding or subtracting randomly generated numbers from the output. The range of the random numbers is defined manually and represents the percentage of exploration. The exploration is high if the range of random numbers covers the value range of the output elements. In comparison to the previous example, this exploration is constant. Every output of the NN is changed to a certain degree. The main intention of the agent is still considered and, to some extend, manipulated. In the long run, the agent will adapt its output in such a way that even the additions and subtractions are not enough to change the intent. By this, the agent will overcome the uncertainties of exploration and maximize the reward.

One of the last parameters to define is the duration of the training. This parameter defines how many training episodes are run through. This can be as well defined as a time limit. The learning process will run until the time limit is reached. This is helpful if a deadline for the results is set and it's not possible to calculate the needed iterations to fill the time-frame. In most applications, the training is aborted after the agent manages to reach a desired average reward over a set amount of iterations. The same can be applied to the manufacturing process. The agent and its NN can be transferred into production as soon as the results show that the agent can improve efficiency by a certain margin.

3.5 Conceptional Implementation and Execution

The most used programming language for the implementation is Python due to the multiple free open-scours libraries for ML. Other languages like C++ or Java are possible as well but lack the supportive libraries. The libraries contain premade algorithms so that the user does not need to implement the mathematical functions manually. They serve as a construction kit to simplify the implementation. The necessary parts and functions like layers and backpropagation can be easily connected like prefabricated building blocks. The most prevalent libraries for implementing ML are TensorFlow [107], PyTorch [108] and Keras [109]. The basic principles are the same in all libraries. The differences between those are not analyzed in this work. The agent and NN, as well as the environment, can be implemented object-oriented. This allows for easy function calling but is not mandatory.

For training, two nested loops are defined. Figure 3.5 shows the general structure of the training process. The inner loop (orange arrows, [A] in Figure 3.5) is for the interactions of the agent with the environment. Information from the simulation or digital twin is received and, after preparation, fed to the NN. The starting state for every training run has to be similar to the actual starting state in the manufacturing process. If the agent is responsible for controlling the manufacturing process as soon as an error or delay occurs, the starting state in training has to be similar. The more the environment differs in training from the physical target use, the more time is wasted to adapt to new starting states. This is why it is important to have an accurate environment. If the environment is modeled too complex, the agent needs more time to solve the problem. If the environment is modeled too simple, the knowledge of the agent is not enough to control the physical manufacturing process. The return of the NN is interpreted, executed and rated. All changing variables are saved to the buffer. To eliminate the need for repeated information request and state generation, the state that is queried after the execution of the agent's action can be used as the input for the NN for the next loop. This loop is repeated until the maximum set number of iterations

is reached or the problem is solved. After one episode is over, the NN is optimized. It is possible to optimize the NN while the episode is still running or after multiple episodes.

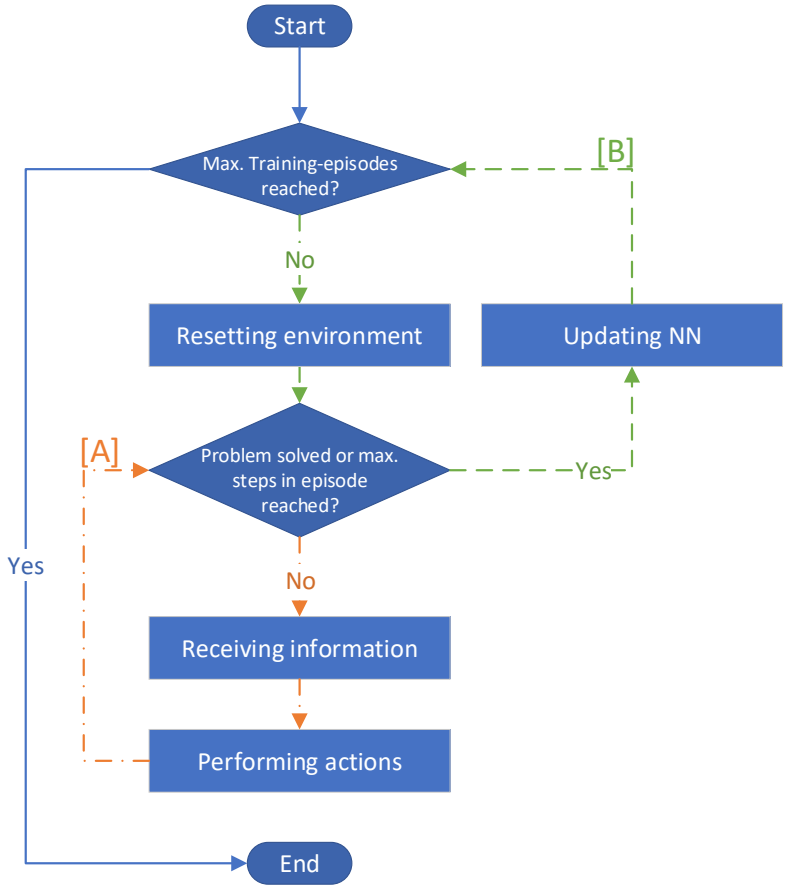


Figure 3.5: Two nested loops for continuous training

The outer loop (green arrows, [B] in Figure 3.5) is for repeating this cycle. The training can be aborted if a certain average reward is reached or the time of training runs out. If the cycle is aborted manually or the cycle stops for some other reason, the NN and its knowledge is lost. All program variables and the NN are only saved in temporary storage. To implement RL in production control, the NN has to be saved after training to an accessible part of the computer storage. To reduce the risk of information loss, the NN can be saved every set number of iterations. This process is supported by further predefined functions in the libraries. To use the knowledge of a pre-trained NN, it has to be loaded from where it is stored. This is done as well by predefined functions that are implemented in the libraries. By this, one NN can be duplicated and used on multiple manufacturing processes if the environments are comparable. Further, it allows easy switching of NNs in production. Multiple agents can be trained on different objectives and exchanged if necessary.

4 Development and Validation

The following chapter gives a detailed presentation about the development of the simulation, fine-tuning of parameters and validation of the agent's interaction.

4.1 Assumptions and requirements for the simulation

The following simplifications and assumptions are considered to reduce the complexity of the simulated manufacturing system. Each product has certain process steps that have to be completed before the product is regarded as finished. Those process steps that are defined by the routing (see Chapter 2.1) must be performed in sequential order. For example, it is not possible to start the third process step before the first and the second one are not labeled as done. Not every product must have the same required process steps as the simulation can produce a variety of products. If steps are not required, they are considered as done. Process steps can only be performed by the machines having the respective necessary skills. Products are not bound to specific machines for their required steps. Multiple machines can perform a variety of process steps. These multi-skilled machines perform their skill with different efficiencies. Every machine is designed to perform one process step with the highest speed compared to others. The processing times at machines that are capable of performing the same process step as a specialized machine, are set higher for that process step than the one at the specialized machine. The machines are subject to unforeseen failure. The probability of failure is the same for every machine and set to a chosen constant. In case of failure, the time to repair is known. Machine failure is only possible if the machine is empty. If a machine started a processing step, it is not possible to abort. The same applies to the transportation of the products. If a product is in transit, it can not be stopped or redirected. Every product can be transported to every machine without limitations.

4.2 Implementation of the Variables and Simulation

The simulation of the specified cell-based manufacturing process is implemented in the programming language Python for ease of understanding and to make full use of the available libraries. It is divided into two parts. The first part (Chapter 4.2.1) is the instantiation of all necessary variables. The second part (Chapter 4.2.2) is the development of a simulation that can work with the created variables. Selected variables will be used for generating the state-vector.

4.2.1 Implementation of the Variables

The first data structure is a matrix named **WorkingTime** (see (4.1)). It is responsible for containing all the information about which machine is able to perform which process step. Each row is assigned to one machine. The columns represent the processing steps (PSs). If the value at a position is not "None", the machine can perform the PS in as many time units as the value at this entry. The matrix can be generated randomly or hard coded manually. The following example shows the manually defined WorkingTime-matrix used for later validation. Access to the values is structured according to mathematical rules. Arrays start at index 0. WorkingTime[3][2] returns the working time of Machine 3 performing PS 2. In this case, the return is the value 10. It is clearly visible that not every machine is capable of performing every PS. By manual definition, the following machine in the matrix is capable of performing the previous step with less than half the efficiency and thus takes longer than double the time. It is important to note, not to think of those machines in a linear way. The machines can be positioned anywhere in the manufacturing plant and have skills completely unrelated to their name. Machines are labeled from "Machine 0" to "Machine 4" for easy structuring in the matrix. The required working time and extra skills do not have to be structured in a diagonal way. In this case, a diagonal layout of the matrix is chosen to easily understand the decisions of the agent for a detailed analysis.

$$\mathbf{WorkingTime} = \begin{array}{c} \begin{matrix} \text{PS 0} & \text{PS 1} & \text{PS 2} & \text{PS 3} & \text{PS 4} \end{matrix} \\ \begin{bmatrix} 2 & 5 & None & None & None \\ 10 & 2 & 4 & None & None \\ None & 10 & 2 & 4 & None \\ None & None & 10 & 2 & 4 \\ None & None & None & 5 & 2 \end{bmatrix} \end{array} \begin{matrix} \\ \text{Machine 0} \\ \text{Machine 1} \\ \text{Machine 2} \\ \text{Machine 3} \\ \text{Machine 4} \end{matrix} \tag{4.1}$$

The next matrix, called the **ProductDesign** (see (4.2)), defines what PSs are needed for each product. The rows are assigned to products and the columns to PSs. The value is either 1 or "None". 1 indicates that this product requires this process step. "None" indicates that this step is not necessary or already performed. Every individual status of the products at this very moment is encoded in this matrix. In this example, product 0 requires all PSs except PS 2. This matrix can be defined manually or be generated randomly at the beginning of every episode.

$$\begin{array}{c} \\ \mathbf{ProductDesign} = \end{array} \begin{array}{c} \begin{array}{ccccc} \text{PS 0} & \text{PS 1} & \text{PS 2} & \text{PS 3} & \text{PS 4} \end{array} \\ \begin{bmatrix} 1 & 1 & None & 1 & 1 \\ 1 & 1 & 1 & 1 & None \\ 1 & 1 & 1 & None & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & None \end{bmatrix} \end{array} \begin{array}{l} \\ \text{Product 0} \\ \text{Product 1} \\ \text{Product 2} \\ \text{Product 3} \\ \text{Product 4} \end{array} \tag{4.2}$$

The **TransportationTime**-matrix (see (4.3)) is created to encode all transportation times from machine to machine for every product. Each row and column is assigned to a machine. The row index is the source machine (SM) and the column index represents the target machine (TM). The value indicated how many time-units are necessary to reach the TM from the SM. As there is no Transportation-time from one machine to itself, the value, in this case, is set to "None". In this case, the transportation time for every product is equal. If necessary, transportation time can be set for every product individually.

$$\begin{array}{c} \\ \mathbf{TransportationTime} = \end{array} \begin{array}{c} \begin{array}{ccccc} \text{TM 0} & \text{TM 1} & \text{TM 2} & \text{TM 3} & \text{TM 4} \end{array} \\ \begin{bmatrix} None & 2 & 4 & 4 & 4 \\ 1 & None & 2 & 4 & 4 \\ 4 & 1 & None & 2 & 4 \\ 4 & 4 & 1 & None & 2 \\ 4 & 4 & 4 & 1 & None \end{bmatrix} \end{array} \begin{array}{l} \\ \text{SM 0} \\ \text{SM 1} \\ \text{SM 2} \\ \text{SM 3} \\ \text{SM 4} \end{array} \tag{4.3}$$

Each machine has an unlimited buffer for awaiting and already processed products. A array called **ProductBucket** (see (4.4)) has one index dedicated to every product. This array saves the position of products that are not moving in the factory. The index is associated with a product and the value with its current position. If a product is in transit or being worked

on by a machine, the value is set to "None". In the following example, products 2 to 4 are at machine 4, product 1 at machine 2 and product 0 is in transit or being worked on.

$$\mathbf{ProductBucket} = \begin{matrix} \text{Prod. 0} & \text{Prod. 1} & \text{Prod. 2} & \text{Prod. 3} & \text{Prod. 4} \end{matrix} \\ \begin{bmatrix} None & 2 & 4 & 4 & 4 \end{bmatrix} \tag{4.4}$$

The matrix **RemainingWorkingTime** (see (4.5)) is responsible for indicating what machine is currently performing which PS on which product. Every row is associated with one machine. The first column contains the index of the product that is being processed at this moment. The second column contains the remaining working time (RWT) for the performed PS and the third what PS is currently being performed. The third column is needed to differentiate what skill of the machines is being used. In the following example, machine 2 is working on product 5 and performing PS 2. The PS will be completed in 8 time-units. "None" indicates that no PS is currently being performed.

$$\mathbf{RemainingWorkingTime} = \begin{matrix} \text{Working on product} & \text{RWT} & \text{performed PS} \end{matrix} \\ \begin{bmatrix} None & None & None \\ None & None & None \\ 5 & 8 & 2 \\ None & None & None \\ None & None & None \end{bmatrix} \begin{matrix} \text{Machine 0} \\ \text{Machine 1} \\ \text{Machine 2} \\ \text{Machine 3} \\ \text{Machine 4} \end{matrix} \tag{4.5}$$

To track the products in transit, a matrix called **EstimatedTimeOfArrival** (see (4.6)) is created. This matrix has as many rows as there are products. Each row is associated with one product. The first one of the two columns is the TM and the second column, the as of now, required time to reach it. The example shows product 0 in transit. It will arrive at machine 2 in 4 time-units.

$$\mathbf{EstimatedTimeOfArrival} = \begin{matrix} \text{TM} & \text{Time to arrival} \end{matrix} \\ \begin{bmatrix} 2 & 4 \\ None & None \\ None & None \\ None & None \\ None & None \end{bmatrix} \begin{matrix} \text{Product 0} \\ \text{Product 1} \\ \text{Product 2} \\ \text{Product 3} \\ \text{Product 4} \end{matrix} \tag{4.6}$$

The last array is the **MachineFailureCounter** (see (4.7)). This array has the same length as there are machines. Each index of the array is assigned to one machine. If the value is set to "None" the machine is operational and working correctly. If a machine fails, the value is changed to the necessary time to repair. The following example shows what the array looks like if Machine 4 is nonoperational for 50 time-units.

$$\mathbf{MachineFailureCounter} = \begin{matrix} \text{Mach. 0} & \text{Mach. 1} & \text{Mach. 2} & \text{Mach. 3} & \text{Mach. 4} \\ \left[\; None \quad None \quad None \quad None \quad 50 \; \right] \end{matrix} \tag{4.7}$$

4.2.2 Development of the Simulation

The second part is the development of all the functions that can work with the created vectors and matrices to simulate a cell-based manufacturing. The flowchart in figure 4.1 visualizes how the simulation is working without the agent. In this example, it is assumed that the actions for every product is generated manually or randomly. At first, all the necessary variables are created (see Chapter 4.2.1). It is assumed that the agent will not be requested to take over the manufacturing process if all products are finished and no work needs to be done. Therefore it is not checked if all products are done before requesting a handling instruction. As soon as an action is given, it is executed. If all PSs of all products are done, the simulation ends. Otherwise, a new action is generated and executed. This cycle is repeated until all PSs on all products are done.

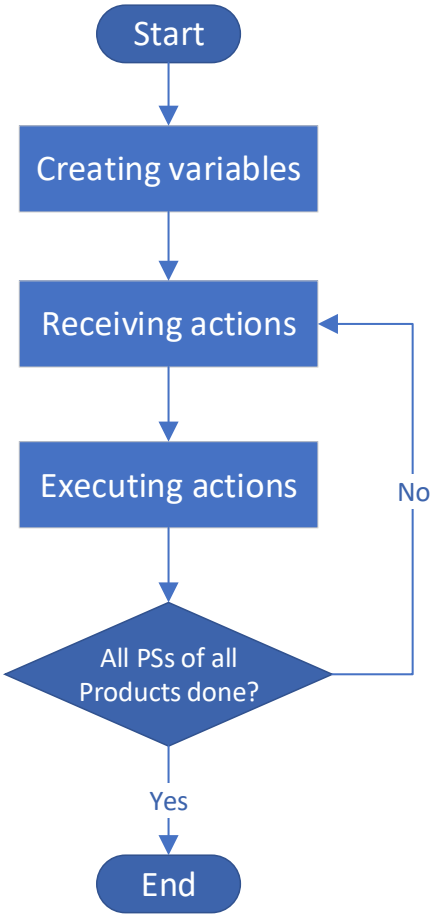


Figure 4.1: General Principle of a Factory Simulation

The process of "Executing actions" is subdivided into further actions. Figure 4.2 shows how it is structured. The first three subprocesses in figure 4.2 are executed independently of the received actions. One pass of this cycle is regarded as one time-unit passed.
At first, every machine that is occupied is performing one time-unit of work on the PS of the product that is currently inside of it. This function requires the RemainigWorkingTime-matrix as an input. Every value in the second column is reduced by one to indicate that the remaining working time on this product for this PS is decreased. If the value is set to "None" no subtraction is carried out as the machine is unoccupied and no value-adding work is performed.

After the processing of one time-unit is performed by the occupied machines, the products of whose PS is finished are ejected out of the machines. From the RemainingWorkingTime-matrix (see (4.5)), every row with a zero in the second column is selected. From the first

column, the product is identified whose PS is finished. The third column contains the PS. With this information, the ProductDesign-matrix (see (4.2)) is changed at the matching position to indicate the completion of one PS. By setting all values of the selected rows of the RemainingWorkingTime-matrix (see (4.5)) to "None" and changing the value of the appropriate ProductBucket-elements (see (4.4)) of the products, to the machine where the PS was performed, the products are ejected.

The information about the transportation of the products is contained in the EstimatedTimeOfArrival-matrix (see (4.6)). In this case, every value in the second column is decreased by one if it is not "None". This subtraction is equivalent to the passing of one time-unit in transit. After the subtraction, it is checked if any of the just reduced values turned to zero. A zero in the second column indicates that a product has reached its TM. The TM is encoded in the first column. The index of the row is associated with the product. This information is used to change the value of the ProductBucked vector at the index of the product to the TM. After that, both elements in the row of those products in the EstimatedTimeOfArrival-matrix (see (4.6)) are set back to "None".

The **action-vector** is a vector with as many elements as there are products. At first, the actions are executed in which a product is requested to be transported to a machine. The actions for this process are encoded as integers from 0 to 4 and represent the TMs. For example, the value 0 at the first element is equivalent to the command of sending product 1 to machine 0. This is only possible if the product that the action applies to is not moving or being worked on. If this is not the case, the action is ignored. The necessary information for the execution is extracted from the ProductBucked-vector. A product can be sent to a machine if the value at the corresponding index at the ProductBucket-vector (see (4.4)) is not "None". This value represents the current position of the product and is used to find out the time for transportation to the TM. The necessary time for transportation from the SM to the TM is encoded in the TransportationTime-matrix (see (4.3)). The TM is directly given by the action. If the SM and TM are not the same, the corresponding value in the ProductBucket-vector (see (4.4)) is set to "None" and the time that is extracted from the TransportationTime-matrix (see (4.3)), together with the TM, are saved in the EstimatedTimeOfArrival-matrix (see (4.6)) in the corresponding row of the product.

As machines are subject to failure, the next step is simulating this property. At first, it is checked if any machine is currently nonoperational and if so, the appropriate value in the MachineFailureCounter (see (4.7)) is reduced by one. As in the transportation-functions, this subtraction indicates the passing of one time-unit. If the value reaches zero, the machine is back operational and the value is set back to "None". The failure is simulated by generating a random number in the range from 0 to 1 and comparing it to the set failure probability. If

the randomly generated number is smaller than the set failure probability and the machine is operational as well as empty, a randomly generated number in a predefined range is set as the new value in the MachineFailureCounter-vector (see (4.7)). If a machine is empty or not, is encoded in the RemainingWorkingTime-matrix (see (4.5)). This procedure is done in order to encode how long the machine will be nonoperational. To eliminate the actual functionality of the machine, the respective skills of the failed machine are copied from the WorkingTime-matrix to a separate matrix and afterward changed to "None" in the WorkingTime-matrix (see (4.1)). By changing the WorkingTime-matrix (see (4.1)), a product can no longer be injected into the failed machine as it does not have any skills at that moment. The vector of the copied skills of the machine comes into use as soon as the value in the MachineFailureCounter (see (4.7)) reaches zero. After that, the copied vector with the original machine skills is copied back into the WorkingTime-matrix. This process is repeated for every machine in every time-step.

The last step in the execution of actions is the injection of products into machines. These actions are encoded as a value of -1. If a product receives the action to be injected into a machine, it is checked if the product is stationary with the help of the ProductBucket-vector (see (4.4)). Secondly, it is evaluated if the machine is empty with the help of the RemainingWorkingTime-matrix (see (4.5)). Further, it is analyzed which next PS the product requires. This required step is compared to the skills of the machine where the product is currently located. If the machine has the skill to perform the product's next step, is operational and empty, the injection is possible. The value of the product index is changed to "None" in the ProductBucked-vector. Elements of the RemainingWorkingTime-matrix (see (4.5)) are changed as well. The required working-time is selected from the WorkingTime-matrix (see (4.1)) and copied to the RemainingWorkigTime-matrix. The index of the product and the performed step are saved in the first and third columns.

The actions are performed in the described order. At first, all actions are executed that are responsible for transporting the products. After that, the action of injecting products into machines. Products are prioritized by their index. This means that the action of transportation and injection of a product which is listed first in the ProductDesign-matrix (see (4.2)) are executed first. In the case of transportation, this does not make a difference as the products do not cross-influence their ability to reach the TMs. This prioritization finds use in the injection of products in machines. If multiple products are ordered to be injected into the same machine, assuming it is possible for all the products, only the highest value one is executing this command. The others can not perform this command as the machine will already be in use by the highest value one. The agent is not informed about this prioritization and has to learn this dynamic by exploring.

The action-vector in (4.8) gives an example of how the action-vector is implemented. Index n of the action-vector is assigned to Product n. The value x is equivalent to sending the corresponding product to Machine x. The value -1 injects the associated product into a machine.

$$\begin{bmatrix} -1 & 2 & 4 & 3 & 0 \end{bmatrix} \tag{4.8}$$

Figure 4.2 gives a visual explanation of how the passing of one time-unit is structured in the simulation. This order is necessary to ensure that a product can be ejected from a machine and instantly receive the order of transportation. The same principle applies to a product that can be injected into a machine without an idle period of time. Further, machine failures are set before the injection of products as the machines can only fail while they are empty. If the injection of products is set before the machine failures, it is possible to evade downtime by constantly keeping a product in the machine.

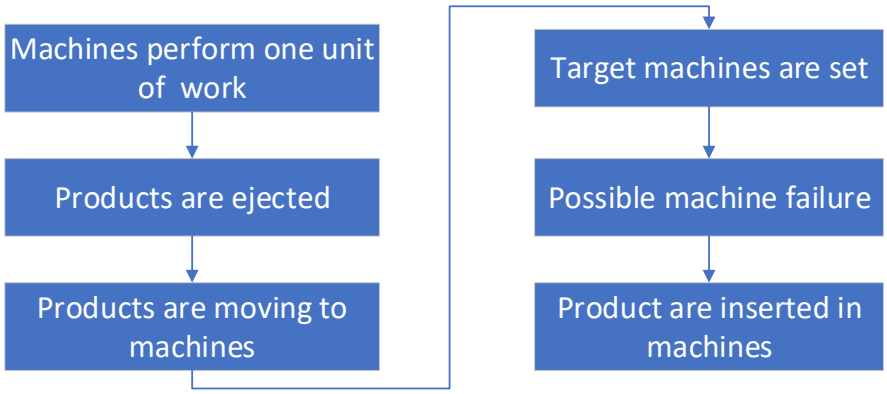


Figure 4.2: Order of Execution of Actions is the Factory-Simulation

4.3 Processing of the Required Data for an MDP

This implementation is focused on providing proof of concept for implementing an RL-based production-control-system in a cell-based manufacturing environment. Therefore, the factory itself is not modeled after one specific factory in the real world. The agent is trained only on one instance of such manufacturing environment. This is evident in the information that is used to generate the Markov-state. Only the elements that can change over time are required. This includes everything except the WorkingTime (see (4.1)) and TransportationTime-matrix

(see (4.3)). Both matrices are not changing for the duration of one episode or the whole training. Constants like the failure probability of every machine is set to a constant value and therefore excluded as well.

The Markov-state is constructed from the five elements shown in table 4.1.

Table 4.1: Required Elements for Construction of the Markov-state

1) ProductDesign (see (4.2))
2) ProductBucket (see (4.4))
3) EstimatedTimeOfArrival (see (4.6))
4) RemainingWorkingTime (see (4.5))
5) MachineFailureCounter (see (4.7))

Figure 4.3 shows the order of the elements from table 4.1 in the state-vector. The

Figure 4.3: Composition of the State-Vector

WorkingTime-matrix (see (4.1)) is the same as in the example in Chapter 4.2.1. At the beginning of every episode, the ProductDesign-matrix (see (4.2)) is completely filled with ones. To reduce feature loss, all values from the MachineFailureCounter (see (4.7)) are divided by 5. This manual prescaling brings the high values closer to the values of the other elements and thus does not suppress the low numbers (see Chapter 3.2.2). This manually chosen divisor requires fine-tuning as it is not helpful to suppress this important information too much.

Prescaling of every vector and matrix individually, as described in Chapter 3.2.2 does not result in the successful learning of the agent. This is possibly due to the loss of importance by prescaling. If the failure-time is brought down to the average value by scaling, the NN takes more time to learn the importance of this information. By keeping the values of the MachineFailureCounter (see (4.7)) higher than the average, more importance is given to them.

After the division of all MachineFailureCounter-elements (see (4.7)), all elements are restructured into a vector and combined into one single vector. This vector currently holds numbers as well as “None” type elements. In the next step, those “None” type elements are replaced

by a -1. This is mandatory as the NN only accepts scalars and the value -1 is not used by the simulation. Lastly, the data-type of the vector is changed to float64 to increase precision and the Min-Max-Scaler is applied.

Figure 4.4 shows how the process of generating a state for the NN is structured.

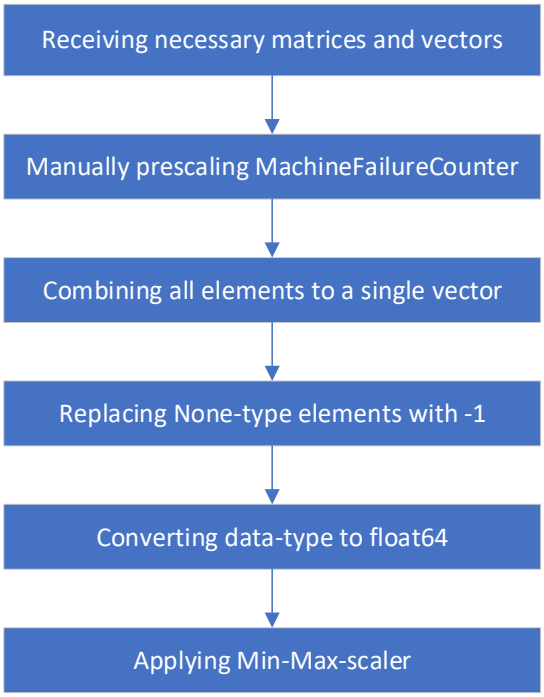


Figure 4.4: Generating a State for the NN

4.4 Required Neurons and Interpretation-Function

The state-vector in Chapter 4.3 is fed into the NN to calculate the optimal actions for the current state. Every element of the vector is given to one input neuron. This means that the number of input-neurons is equal to the length of the state-vector. In this case the ProductDesign-matrix (see (4.2)) contain 25 elements, EstimatedTimeOfArrival (see (4.6)) contains 10, ProductBucket (see (4.4)) and MachineFailureCounter (see (4.7)) contain 5 elements each and RemainingWorkingTime (see (4.5)) contains 15 elements. The necessary number of input-neurons is set to 60. The same principle, as explained in Chapter 3.2.3, is applied to calculate the number of necessary output neurons. The chosen environment contains five products and five machines. Each product has the option of traveling to a

machine, staying in place or being injected into a machine. This results in seven possible options for every product. To reduce the action-space, the idle signal is replaced by the action to travel to the current position of the product. This action can not be executed if the product is already at the TM and thus will not change the position of the product. Because of this property, it can be used as a replacement. With the help of this reduction, the number of required output-neurons is calculated in equation (4.9).

$$\text{Required Output Neurons} = (5 + 1) * 5 = 30 \tag{4.9}$$

To create a action-vector that can be interpreted by the simulation, the returned vector of the NN is subdivided into five equally long sections, because every product has the same freedom of movement. The first section is assigned to the first product from the ProducktDesign matrix. The interpretation function is responsible for finding the highest value in that subsection and the corresponding index. The index is not the index from the whole action-vector but from the subsections. The possible indices range from 0 to 5. From this number, 1 is subtracted to directly encode the TM. All resulting values from 0 to 4 are representing TMs. If the highest value is in the zeroth element of the subsection, its index is 0. After subtraction, the -1 is representing the order of injection into a machine where the product is located. All the extracted orders are compiled into a 5-element vector and forwarded to the simulation. The simulation is only executing valid actions. Figure 4.5 shows how the Interpretation-function is extracting the action from the raw output of the NN.

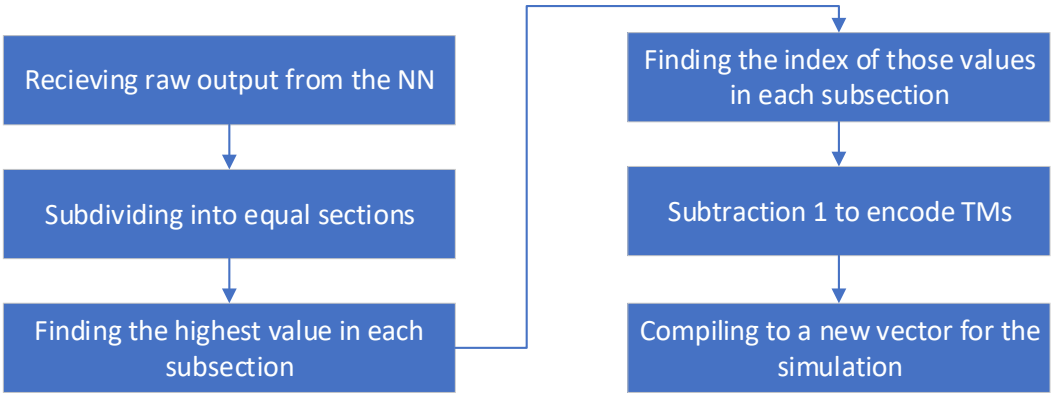


Figure 4.5: Process of Extracting actions from the raw output of the NN

If two products receive conflicting action-signals, the product ranked higher in the ProductDesign-matrix (see (4.2)) is prioritized. This prioritization is encoded in the simulation and is not included in the state-vector (see Chapter 4.2.2).

4.5 Implementation of the Neural Net

After the number of input and output neurons for the agent is set, the next step is to decide how the hidden layers of the NN are structured. The NN of the critic is designed similarly to the NN of the actor. This task requires extensive testing. For the implementation of the NN, the library PyTorch is being used to construct the NN and all accompanying functions. After some testing, the decision is to use a Feed-Forward-NN (see Chapter 3.3b) for the design of the NNs of the actor and critic. The NN of the actor consists of four layers. The first layer that receives the input is connected to the second layer with 256 neurons. All the layers are fully connected. The third layer has 128 neurons and is connected to the fourth layer, which is the output-layer. Figure 4.6 gives a visual representation of the NN of the actor. Each node is representing approximately 30 neurons.

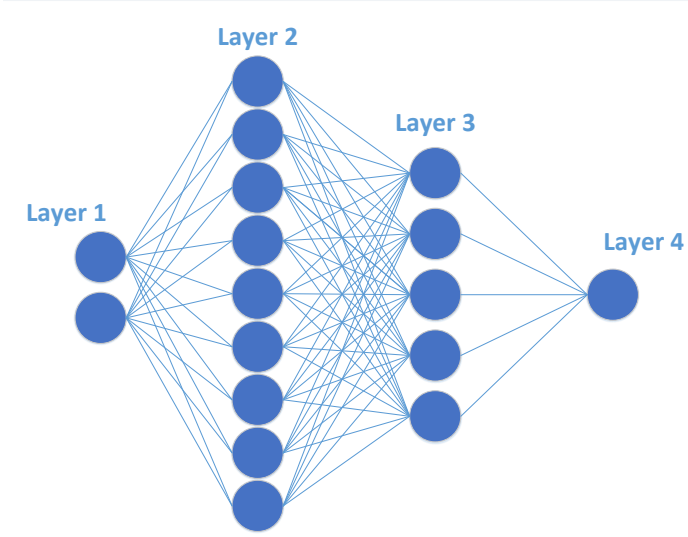


Figure 4.6: Layer-Structure of the NN of the Actor

The NN of the critic has on the input layer as many neurons as the sum of the number of input and output neurons of the actor. The second layer consists of 256 neurons, which are connected to the third layer with 256 neurons. The fourth layer contains 128 neurons and the last layer consists out of 1 neuron. All layers of the critic are fully connected (see Chapter 3.3.1).

In the next step, it is to define the activation-function (see Chapter 2.3) of each layer. All layers except the last one of the critic are set to ReLU-functions. The neuron of the last layer returns the true value of the combined inputs from the third layer without applying an activation function.

All layers except the last one on the actor are set to ReLU as well. The activation function of the last layer is set to tanh (see Figure 3.4b). It is not possible to get sufficient results by setting the activating function of the last layer of the agent to ReLU (see Figure 3.4a). This is possibly due to the disregarding of negative values by the activation function.

4.6 Implementation of the Buffer

The purpose of the buffer (see Chapter 3.3.2) is to save all information of one transition. A transition contains the state before and after the environment is changed, the output from the NN, as well as the received rewards and the "done"-indicator. At the beginning of the training, the buffer is initialized with a maximum number of stored transitions of 5e15. One transition contains 60 elements for the state before and after the actions, 30 output elements from the NN, one reward and the done indicator. This results in a tuple with 5 sub-arrays and 152 elements in total for one transition. If all buffer is filled, the calculated required storage space is shown in equation (4.10).

$$152 * 64 \text{ bit} * 5e15 = 6080000000 \text{ Tb} \tag{4.10}$$

After every execution of actions, a function of the buffer is called that saves the transition. If the buffer is filled up completely, the oldest 20% of the stored transitions in the buffer are deleted. This makes it possible to continue training if the available storage is limited. A further function of the buffer is the sampling of the past transition for the backpropagation. As soon as the agent calls the Sample-Function, a random vector of indices is created. It is possible that some indices are selected multiple times. The indices point to the tuples in the storage. From the selected transitions, five new arrays are created. Each array contains only action, states, rewards or indicators. Those arrays are returned to the agent.

This limiter of elements is set high on purpose to eliminate the risk of Catastophical Forgetting (see Chapter 3.3.2). In all training runs of the experiment, the number of saved transitions in the buffer does not exceed 21e6, which is equivalent to approximately 26 GB.

4.7 Implementation of the Reward-Function

The goal of the agent is to finish all PSs of all products as fast as possible by avoiding faulted machines. Rewards are only given to the agent if the problem is solved or the maximum amount of steps in the episode is reached. The reward function is designed in two parts. The

first part is a reward given to the agent for the completion of PSs. At the end of an episode, the number of finished PSs is cubed to ensure an exponential function (see Chapter 2.4.4) and returned to the agent. Figure 4.7 shows the exponential curve of reward, depending on how many PSs are completed. This distribution is incentivizing the agent to complete as many PSs in the available time frame.

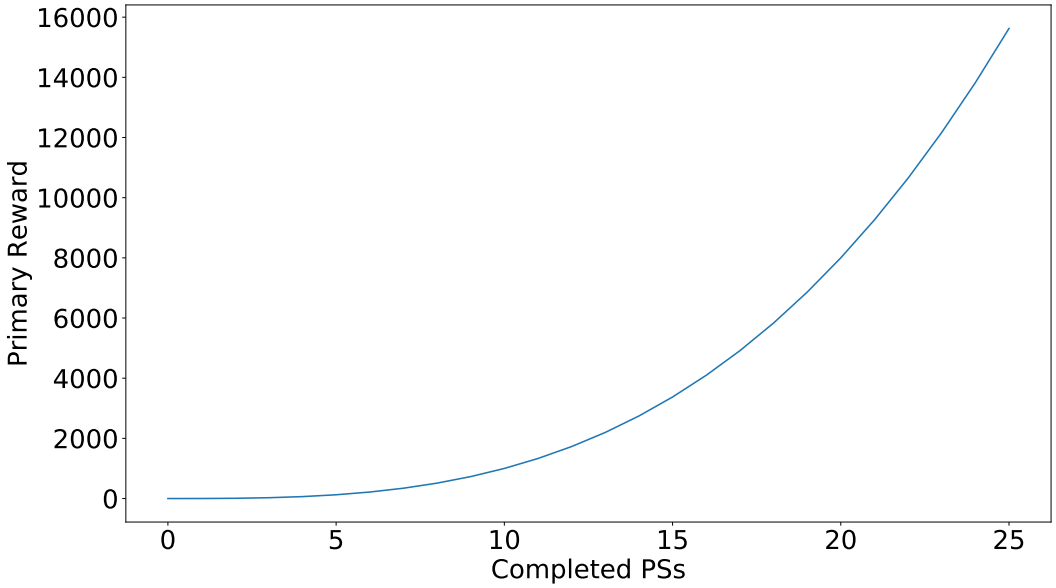


Figure 4.7: Primary Received Reward for completing PSs

The second part is the reward that is given to the agent if the problem is successfully solved faster than the permitted time-frame. Every time-unit that is left after all PSs are completed is cubed and added to the first reward. The sum of both rewards is returned to the agent. By this distribution, the agent is incentivized to perform as many PSs as possible as fast as possible. Figure 4.8 shows how the time-reward is distributed as a function of the time left.

In comparison from figure 4.8 to figure 4.7, it is evident that the time-reward is clearly dominating after the spare time in one episode is over 25 time-units. By this way of distributing rewards, the incentive is set to primarily minimize the necessary time to complete all PSs.

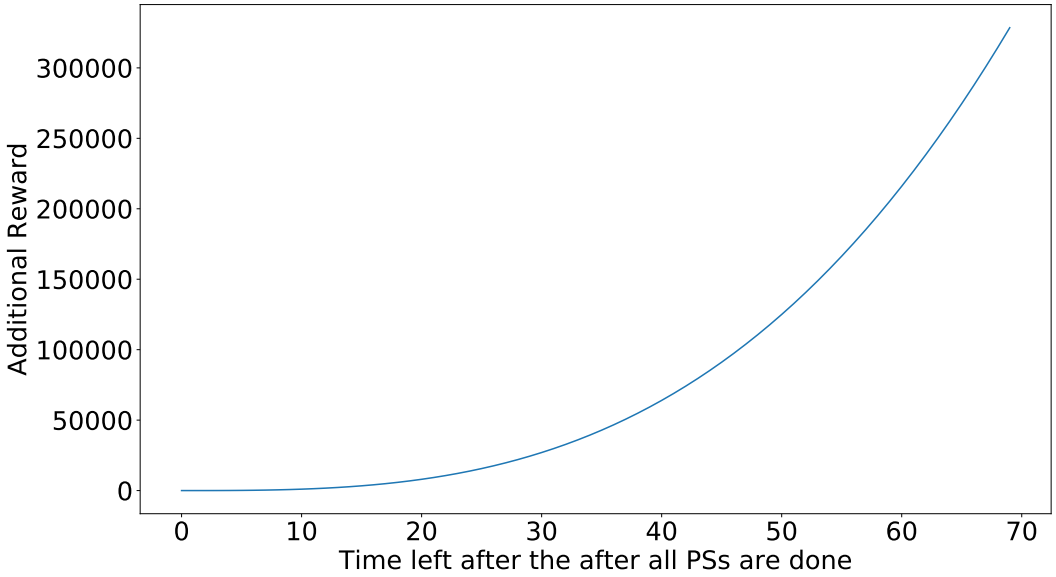


Figure 4.8: Additional Reward as a Function of Time

4.8 Implementation of Exploration

As mentioned in chapter 3.4, it is necessary to implement exploration. This exploration helps the agent to solidify its knowledge on how to act in all states of the environment optimally. Exploration is implemented by adding values to the output-vector of the NN before the indices of the maximum values are extracted, or the output is saved to the buffer. For this, a new vector, from now on called the noise-vector, is created that is as long as the output-vector from the NN and filled with random numbers. The function that generates the random numbers is modeled as a normal distribution with an average of 0. A high standard deviation can lead to greater values that overwrite the values from the NN completely. As the NN is generating values only in the range from -1 to 1, which is defined by the tanh-activation-function (see Chapter 4.5), a high standard deviation leads to constant exploration and no action of the agent. To the exemplary output-vector from the NN shown in (4.11), a noise-vector presented in (4.12) is added that is created with a standard deviation of 0.0001.

$$\begin{bmatrix} 0.005 & 0 & 0.022 & 0.044 & 0.01 & 0.034 & -0.012 & 0.01 & \dots \end{bmatrix} \tag{4.11}$$

$$\begin{bmatrix} 8e\text{-}5 & 7e\text{-}5 & -8e\text{-}5 & -1e\text{-}4 & 1.4e\text{-}4 & -3.6e\text{-}5 & -1.8e\text{-}5 & 2.4e\text{-}5 & \dots \end{bmatrix} \tag{4.12}$$

In this example, the addition of noise does not change the vector enough to cause a different index to be extracted. This kind of exploration shows its effect over many iterations as the randomly generated numbers are not limited to a range and will change the output-vector eventually. This parameter requires fine-tuning for every application and should not be carelessly set to an arbitrary number.

4.9 Summary of Constants and Implementation

In this chapter, a summary of the used constants will be presented. These constants will be tested in multiple test-runs, to analyze which values lead to the best performance of the RL algorithm.

Learning rate = 0.01 - 0.0001
Various learning rates will be tested over the course of over 30 tests.
0.01 is the highest tested one and 0.0001 is the lowest tested. For the final training, the learning rate is set to the value that led to the best results.

Noise standard-deviation = 0.5 - 0.0001
For the generation of random numbers, the standard deviation is tested in a range from 0.5 to 0.0001. This test is to analyze which rate of exploration is leading to the highest long-term reward.

Gamma = 0.99
Gamma is set to 0.99. This is a very typical value for RL problems as most of the time, the overall long-term goal of solving the problem is in focus [110]. Further values of gamma are not tested.

Batch-size = 2000 - 100
The batch-size is tested in the range from 100 to 2000. As with the learning rate and standard deviation, the Batch-size is set to the value that led to the best results.

random-steps-before-takeover = 10
To introduce more dynamic into the system, ten random steps are performed to randomize

the first state the agent observes at the beginning of every episode. This is implemented to train the agent to generalize and not start in the same state every time. During this time, machines can fail and PSs completed.

Failure-Prob = 0.025
Min-Error-Time = 40
Max-Error-Time = 70
The main dynamic of the environment is the element of machine failures. As mentioned in Chapter 4.2.2, machines can only fail when they are not in use. The duration of the failure is set in the range from 40 to 70 time-units. This means that in the case of failure, a machine is not operational for at least half of the duration of the episode. The failure probability is set to 2.5% on every machine for every time-step the machine is not performing work. It is possible for a machine to fail during the execution of the ten randomly performed steps.

max-timesteps = 70
The limit of the maximum interactions of the agent with the simulation in one episode is set to 70. This value is selected as the minimum required time for completing all the PSs of all products ranges from 30 to 40 time-units, depending on the initial position of the products. This minimum required time is calculated by applying a FIFO-scheduling rule with no machine-failures. In addition, one machine is set to perform only its most efficient skill to simulate the current state of a manufacturing environment.

Layer-structure of the Agent = 60-256-128-30
Layer-structure of the Critic = 90-256-256-128-1
The NN of the agent and critic are designed, as mentioned in chapter 4.5. Further NN designs are not extensively tested.

Maximum stored transitions in the buffer= 5e15
The limiter of the maximum elements stored is set to this very high value on purpose. The number of stored transitions in the buffer never exceeded 21e6.

Further parameters are needed for implementing the Actor-Critic approach. In all tests parameter ”policy-noise” is set to the same value as the learning rate. These parameters are taken from [111].
polyak = 0.995, **policy-noise = 0.0001**, **policy-delay = 2**

Figure 4.9 shows the complete process of training the agent on the developed simulation.

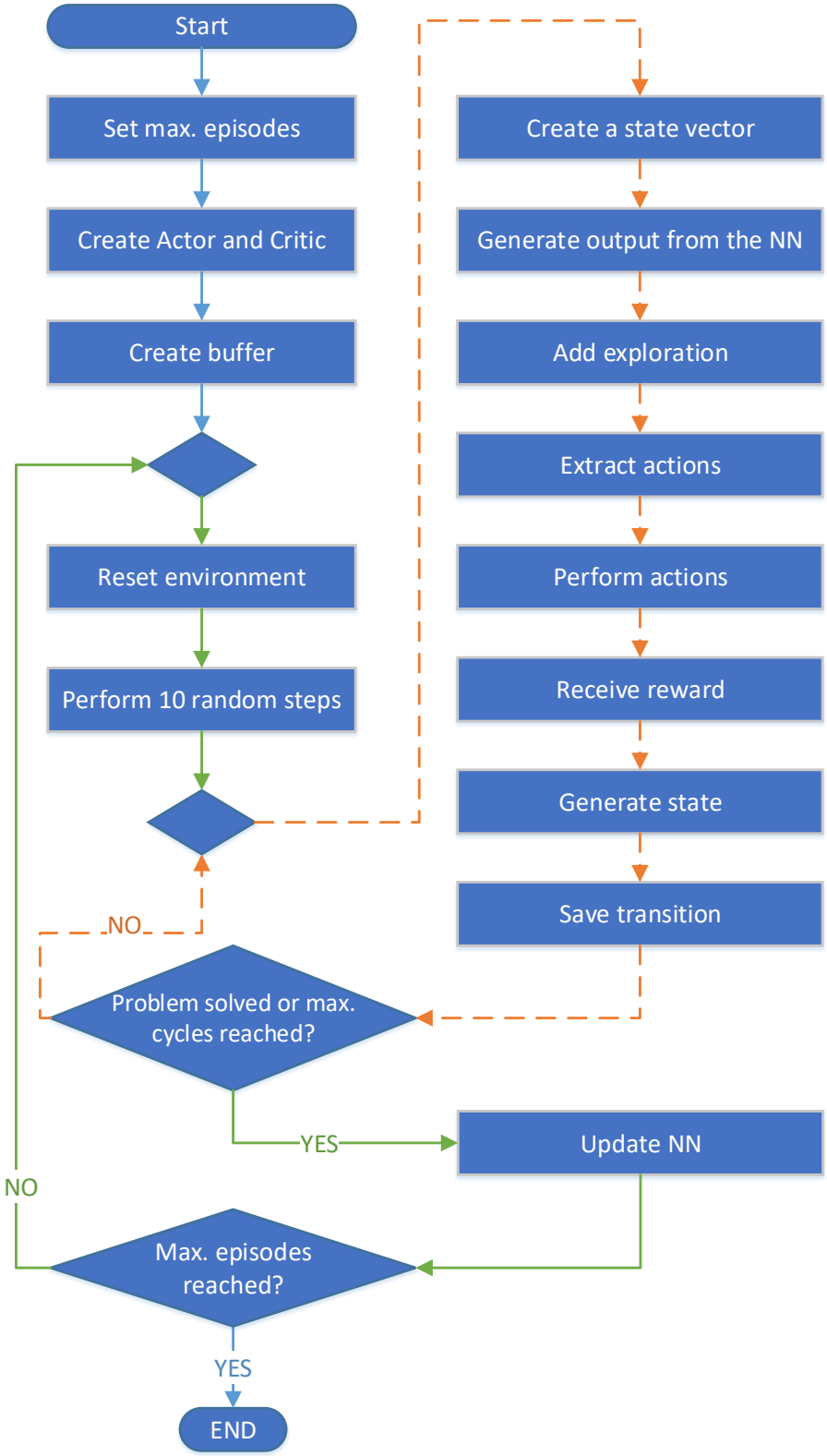


Figure 4.9: Process of training the Agent on the developed Simulation

4.10 Training and Validation of the RL-Approach

In this section, multiple small-scale tests are conducted to select the best possible values for some global constants. The aim of these tests is to analyze the performance of the agent and select the best combination of variables for further testing. In the focus of these tests are the learning rate, the level of exploration and the batch-size.

The first graph in figure 4.10 shows how a scheduling rule is performing without the presence of machine failures. The applied scheduling rule is FIFO. If a PS has been performed, the product moves to the next machine that can perform the next required PS most efficiently. In the case that two products arrive at a machine simultaneously, the prioritization of products is from top to bottom as encoded in the ProductDesign-matrix (see Chapter 4.2.2). The agent is excluded from the simulation when a scheduling rule is applied.

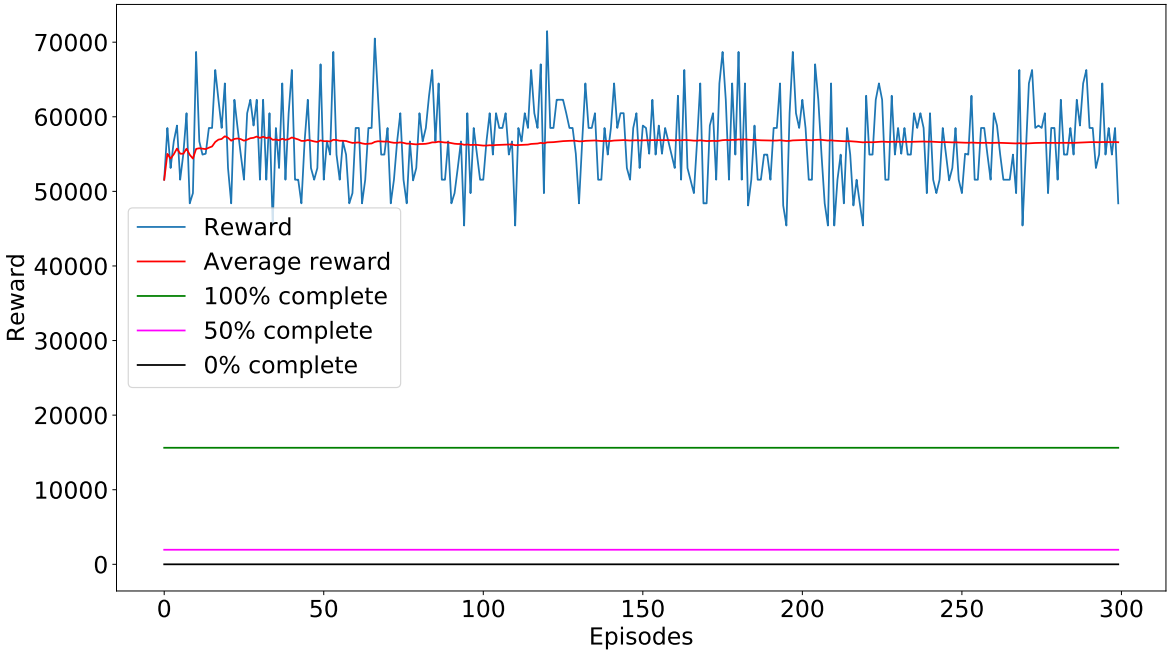


Figure 4.10: Rewards for FIFO-scheduling in a Simulation without Machine Breakdowns

In every episode, the scheduling rule FIFO is able to complete all PSs. This is evident in the fact that each reward, represented by the blue line, is above the green line, which represents the completion of all PSs (100% completed). The crossing of this line is equivalent to completing all PS in less than 70 time-units. The reward of these episodes is, on average, 56300. The average reward is represented by the red line. This reward is subdivided into the PSs-reward and the time-reward.

15625 is the reward for completing all PSs (see Figure 4.7), which leaves 40675 as the time-reward (see Figure 4.8). It takes on average 34 time-steps to complete all PSs. The fluctuation is caused by the random initial position of the products at the beginning, by performing 10 random actions. The graph in figure 4.10 includes horizontal lines (green, magenta and black) that indicate the percentage of completion and the equivalent completion-reward of PSs for better visual representation.

The graph in figure 4.11 shows the success of FIFO-scheduling in the case of machine failure. The failure probability is constant and the duration is set randomly, as described in chapter 4.9. The results of this test show that FIFO is significantly less successful in the simulated environment with machine breakdowns. Unaffected runs by machine breakdowns manage to finish on average in 34 time-units. Affected runs however, do not manage to process all PSs in most cases. The blue graph, which represents the reward of every episode, rarely exceeds the green line, which represents the completion of all PSs (100% complete). This is due to the time it takes for a machine that is necessary for a process step to be back operational. In the worst case, a machine loses its functionality for the duration of the entire run (70 time-units). The average reward of affected and unaffected runs combined is 5200. The average reward of only the affected runs is 3500. This shows the importance of rerouting products to alternative machines in the case of a machine malfunction.

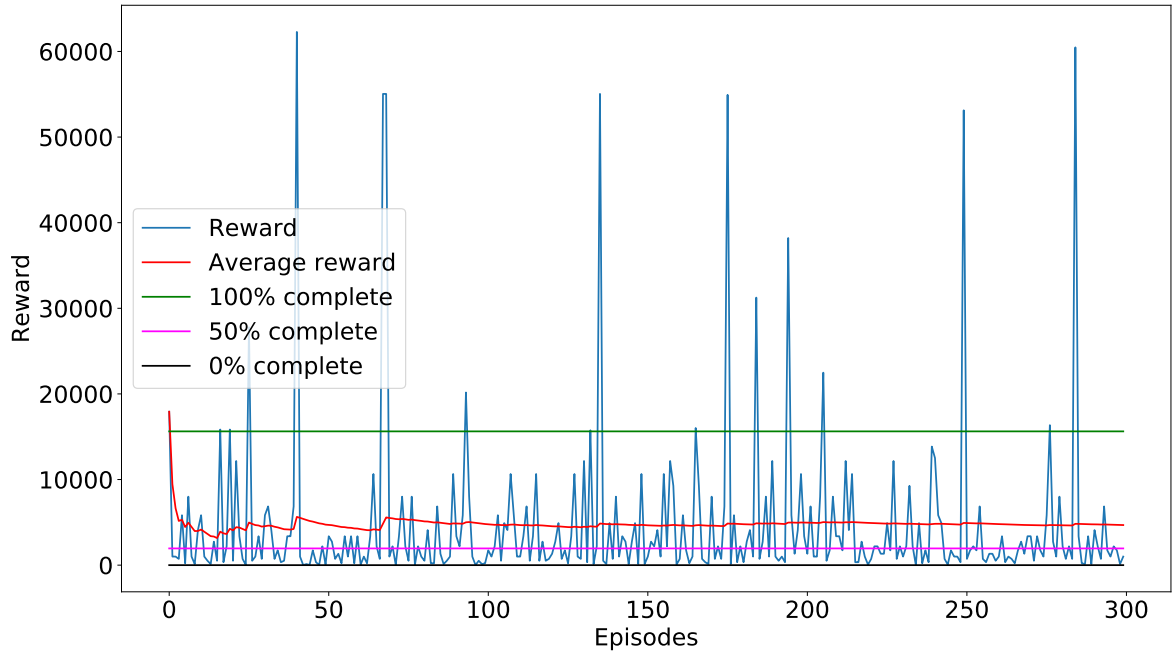


Figure 4.11: Rewards for FIFO-scheduling in a Simulation with Machine-Breakdowns

The FIFO scheduling rule is modified to generate a baseline for the highest possible reward that can be achieved by the agent. With this modification, products are only sent to the most efficient functioning machine for each PSs. If a machine fails, the products from the buffer are rerouted as well. They do not have to wait in the buffer of one machine until it is back operational. By this, the faulted machines are avoided and the most amount of PS is performed.

Figure 4.12 shows that the results of this new rule are visibly better than achieved by the unmodified rule. The average reward is 12700, which is equivalent to the completion of 93% of all PSs in the given time-frame. This scheduling rule is able to complete all PSs more often. This is represented by the blue line crossing the green line more often than in comparison to figure 4.11. The major limiting performance factor is the presence of simultaneous machine breakdowns. In cases where all machines fail that can perform a particular step, no further manufacturing is possible. This especially applies to the first two and last to machines. For example, only machine 0 and machine 1 are capable of performing PS 0. Both of them being operational at the same time impacts the manufacturing process significantly.

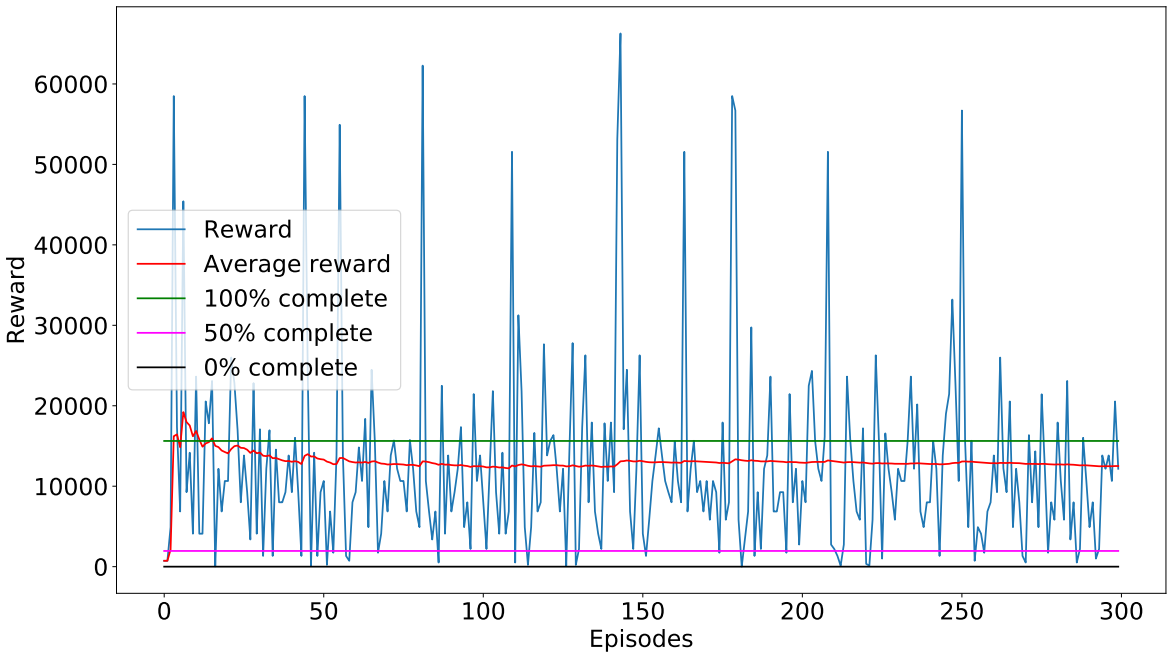


Figure 4.12: Rewards for improved FIFO-scheduling in a Simulation with Machine-Breakdowns

Due to this problem's complexity and the vast amount of parameters, small-scale tests are performed to tune these parameters. For selecting the best possible values to achieve the best results, the following tests have been conducted. The focus of this analysis is Batch-size, Learning-rate and Exploration. The base parameters are set to the values shown in table 4.2.

Table 4.2: Base parameters for Testing of the developed Algorithm

Learning rate = 0.00025
Exploration (Noise standard-deviation) = 0.0001
Gamma = 0.99
Batch-size = 100
Maximum time-steps in one episode = 70
Random steps before the agent takes over = 10
Failure probability = 0.025
Minimum failure duration = 40, Maximum failure duration = 70
Construction of the Agent's NN = 60-256-128-30
Construction of the Critic's NN = 90-256-256-128-1
Maximum transitions in the Buffer= 5e15
polyak = 0.995, policy-noise = 0.0001
noise-clip = 0.3, policy-delay = 2

Starting with these values, one parameter is changed at a time to analyze its effect on the performance of the agent. It is important to point out that these parameters are not cross testes due to time constraints. The duration of the training is set to approximately 30000 episodes for every test. The further development of the results is not analyzed in this small-scale parameter test.
These tests are conducted on the hardware and software shown in table 4.3.

Table 4.3: Used Hardware and Software for Testing and Training

Python: v3.7
PyTorch: v1.4.0
Operating system: Microsoft Windows 10 Pro
CPU: AMD Ryzen 5 3600X, 3800 MHz
GPU: NVIDIA RTX 2080 SUPER
RAM capacity: 32GB

The first conducted test, shown in figure 4.13, analyses the agent's success with different batch-sizes. Analyzed are the batch-sizes 100, 300, 500, 1000 and 2000. Especially noticeable is the agent's lacking success that only samples 100 transitions from the buffer to optimize

the NN. The end-result of the batch-sizes 500, 1000 and 2000 are very similar. The training with a batch-size 2000 has an average gradient in the first two thousand episodes, but the performance deteriorates to an average reward of zero over the next 6000 episodes. After that, the reward quickly rises past the batch-size 100 to the same level as with the batch-size 300, 500 and 1000. The blue graph has the steepest learning curve at the start but is subsequently overtaken by the green and black one. Batch-size 2000 leads to the best performance in these test-runs although the performance in the beginning is unstable.

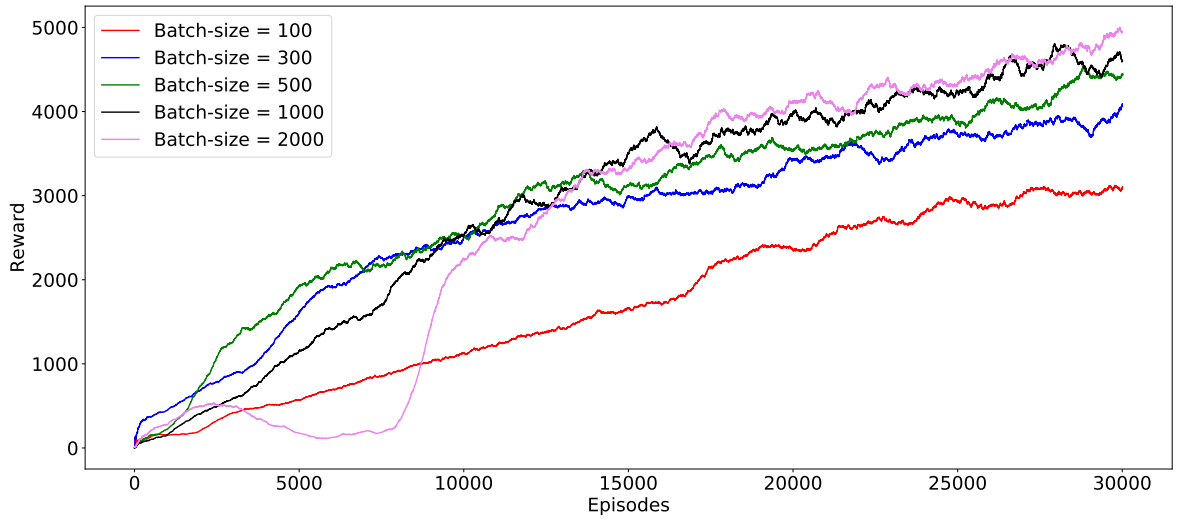


Figure 4.13: The Result of varying Batch-sizes

The second test, shown in figure 4.14, depicts the results of varying the standard deviation in the creation of the noise-vector. Analyzed are the standard deviations: 0.0001, 0.001, 0.01, 0.1 and 1.

The red graph in figure 4.14 is the same as the red graph in the previous test, as they both are the baseline for all the tests (see Figure 4.13). The learning curves of the green, blue and magenta graphs are very similar in their overall gradient. The black graph has a medium gradient in the beginning and is not flattening until episode 16000. Due to its consistency, the black graph manages to achieve significantly better results in the long run. The violet graph representing the standard deviation of 1, achieves visibly better results in the first 5000 episodes compared to others. This progression is anticipated as more exploration will help the agent find better solutions faster at the beginning of the training. In the long run, continuous exploration is not helping the agent to achieve better results.

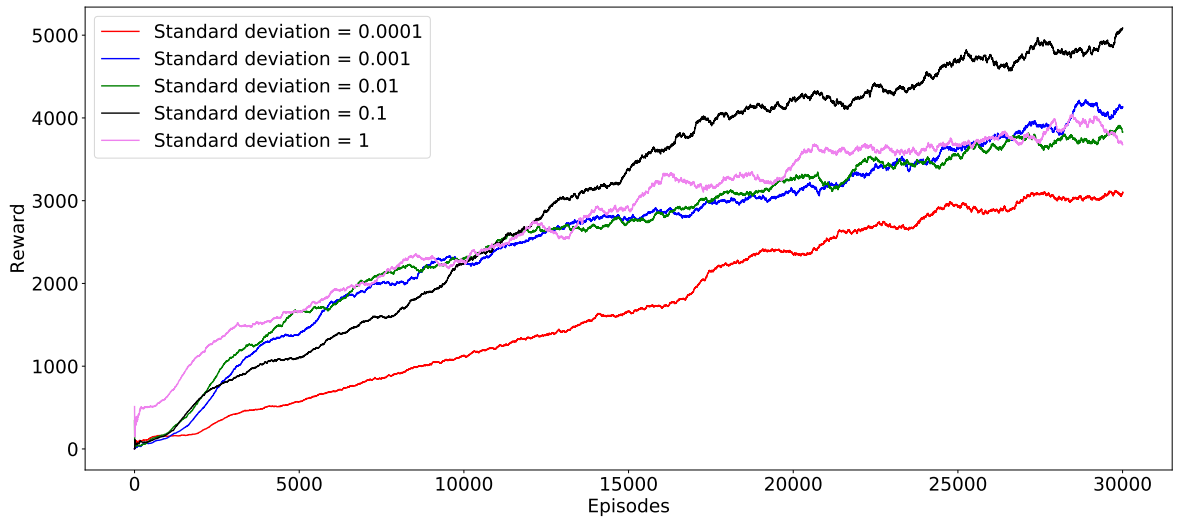


Figure 4.14: The Result of varying Standard Deviations in the Creation of the Noise-Vector

The third test is the analysis of the effects of different learning rates. The following five learning rates are tested: 0.0001, 0.00025, 0.001, 0.003 and 0.01. The graphs in figure 4.15 shows the result of this test.

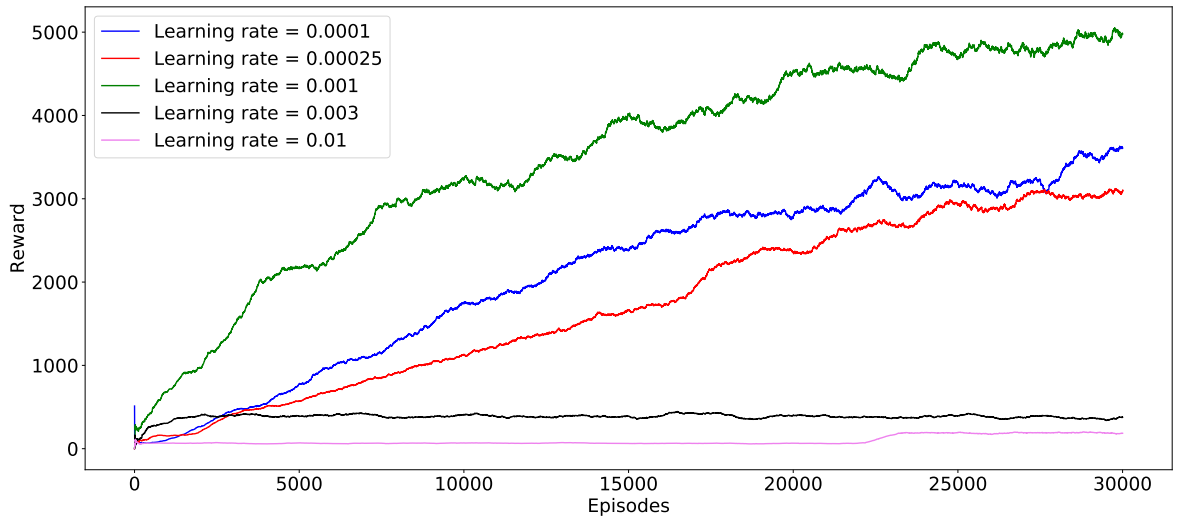


Figure 4.15: The Result of different Learning Rates

The red graph is the baseline for comparison with a learning rate of 0.00025 (see Figure 4.13). The highest learning rate with the value of 0.01 does not achieve any significant results in this test. The agent does not manage to adjust the values of the NN to the right amount

to achieve high positive rewards. Similar insignificant results are achieved by a learning rate of 0.003. The black graph shows that the agent does not consistently receive a reward of more than 500 with this learning rate. The blue graph shows the results of a learning rate of 0.0001. It performs visibly better than with a learning rate of 0.00025, although the relative difference of these learning rates is not extreme. The most successful agent has a learning rate of 0.001, represented by the green graph. It shows that the gradient is consistently steep from the beginning and achieves better results than with any other learning rate modification.

4.11 Detailed Analysis of the Agent's Performance

One extensive training is conducted to analyze the implemented approach's ability to decrease the necessary time it takes to complete all PSs on all products in the simulated environment. Further, it is analyzed if the implemented RL-approach is viable for online-scheduling. Based on the previous tests, the parameters described in table 4.4 are selected for a detailed analysis of the performance of the agent.

Table 4.4: Selected parameters for a in-depth Analysis of the Algorithm's Results

Learning rate = 0.001
Exploration (Noise standard-deviation)= 0.0001
Gamma = 0.99
Batch-size = 100
Maximum time-steps in one episode = 70
Random steps before the agent takes over = 10
Failure probability = 0.025
Minimum failure duration = 40, Maximum failure duration = 70
Construction of the Agent's NN = 60-256-128-30
Construction of the Critic's NN = 90-256-256-128-1
Maximum transitions in the Buffer = 5e15
polyak = 0.995, policy-noise = 0.0001
noise-clip = 0.3, policy-delay = 2

The graph in figure 4.16 shows how the reward is developing over 60000 episodes. The training is conducted on the same hardware as all previous tests (see (4.3)). The red graph presents the average running reward of the last 1000 episodes. Additional horizontal lines are added in figure 4.16 to visualize where the agent hit certain milestones. The yellow line represents the agent's reward for completing 50% of all PSs in the available time-frame. The pink line represents the unmodified FIFO scheduling rule's average success, including affected and

unaffected simulation runs. The black line is the average reward of only affected simulation runs with the primitive FIFO scheduling rule. The orange line represents the agent's reward for completing 75% of all PSs in the available time-frame. The blue line is the average success of the improved FIFO rule. The green horizontal line is the agent's reward for finishing all PSs (100%) in the available time.

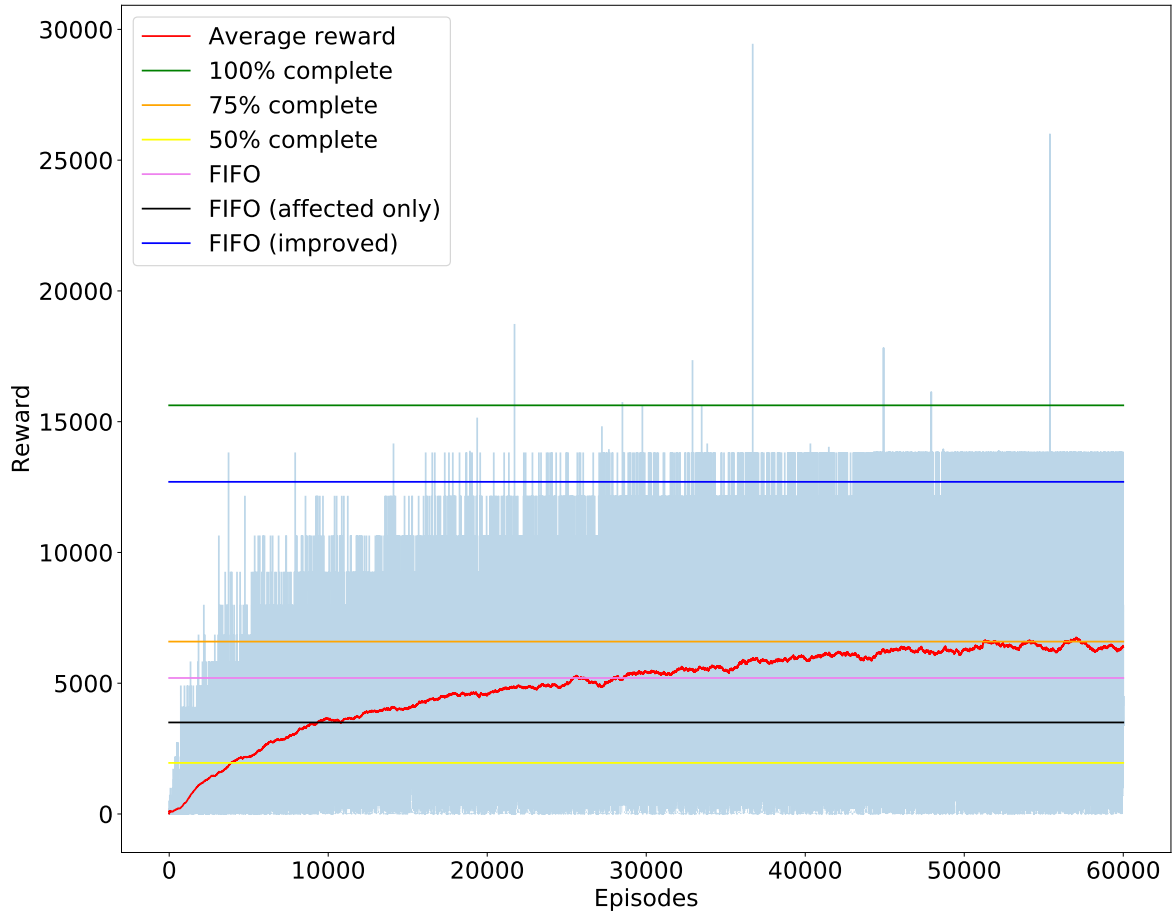


Figure 4.16: Result of the Agent over the course of 60000 episodes

The results of this test, based on the parameters in table 4.4, show that the agent is capable of performing visibly better than the primitive FIFO scheduling rule in the case of machine failures. This goal is reached after approximately 10000 episodes when the average reward represented by the black line is surpassed by the red line in figure 4.16. After episode 30000, the agent is performing on average slightly better than FIFO, including the unaffected runs. This is a clear sign that the agent identified the correlation of machine breakdowns and received rewards. The agent is not able to exploit episodes where no machine failures are present. In the long run, the performance is not as good as the modified FIFO rule that reroutes products based on machine status. The agent is only capable of completing, on average, 75% of all PSs. This equals to a reward of 6430. The gradient of the red learning curve is continuously flattening until episode 50000. After episode 50000, no significant improvement is noticeable. In this test, the agent managed to complete all PSs in nine episodes. The agent does not manage to learn from these events and further improve its performance. The translucent columns in figure 4.16 show the reward for every episode. In the last 10000 episodes, the agent oscillates between a reward ranging from 13824, which is equivalent to completing 24 PSs, and medium four-digit rewards. Some specific values are taken out from the array that stores all rewards for visual understanding (see (4.13)). The agent received these rewards from episode 55555 to episode 55575. The values displayed in bold are the reward for completing 23 or 24 PSs.

[4913, 6859, **13824**, 3375, **12167**, 512, 5832, **12167**, 5832, **13824**,
5832, 5832, 6859, **12167**, 4913, 9261, 5832, 4096, 2744, 1000] (4.13)

The graphs in figure 4.17 shows the distribution of completed PSs in increments of 10000 episodes. In the first 10000 episodes, the distribution is very similar to a normal distribution with an average of 12 completed PSs. Throughout the training, the agent improves its performance and the peak of the distribution is moved closer to the right, which is equivalent to performing more PSs more frequently.

Figure 4.18 compares the distribution of finished PSs in the last 10000 episodes of the training of the agent to the standard and modified FIFO rule. In most cases, the agent is performing 19 to 24 PSs. The distribution of the standard FIFO rule is spread out close to even. It is visible by comparing the blue and green graph in figure 4.18 that the agent cannot exploit episodes where machine failures do not impact the manufacturing process. The standard FIFO rule managed to finish all PSs close to 800 times while the agent does not manage to finish once. The agent's performance is very similar to the modified FIFO rule when only 0 to 10 PS are possible to complete. These events are caused by the failure of multiple

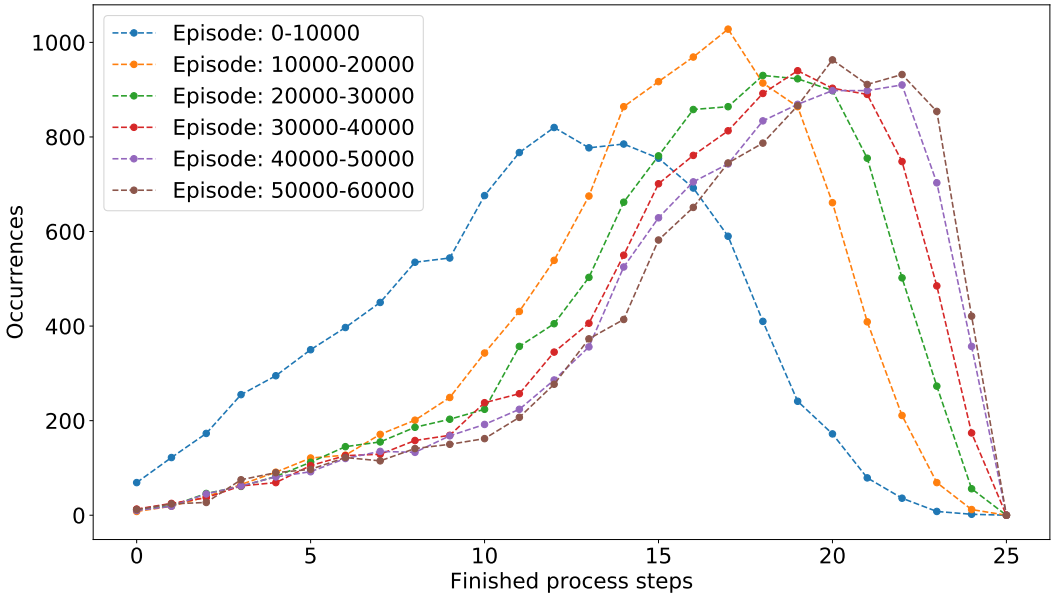


Figure 4.17: Changing Distribution of finished process steps over the course of 10000 episodes

machines simultaneously at the beginning of an episode. This means that the agent is capable of performing close to the mathematical optimum at the beginning of the episode but loses this capability after some steps. Especially noticeable is the lack of success of the agent in completing all PSs in comparison to the modified FIFO rule. Through simple reroutings, the modified FIFO rule is capable of completing all PSs close to 3000 times out of 10000 runs. This behavior can not be replicated by the agent.

To analyze the online-reaction capability of the developed RL-approach, additional functions were added that record the time it takes to execute the implemented functions. The overall time necessary to complete one episode is, on average, 0.6 seconds. This includes resetting the simulation, performing 10 random actions, calculating 70 actions, updating the NN as well as saving the trained NN and reward-history to an external hard-drive. 0.5 seconds are required for updating the NN by back-propagation. This is the most time-intensive task. The time it takes for the agent to calculation one single action is 0.001 seconds. This time is the decisive factor for the implementation of an online-reaction system. In the case of production scheduling, this time can be viewed close to real-time and thus fulfills the required online-reaction capability.

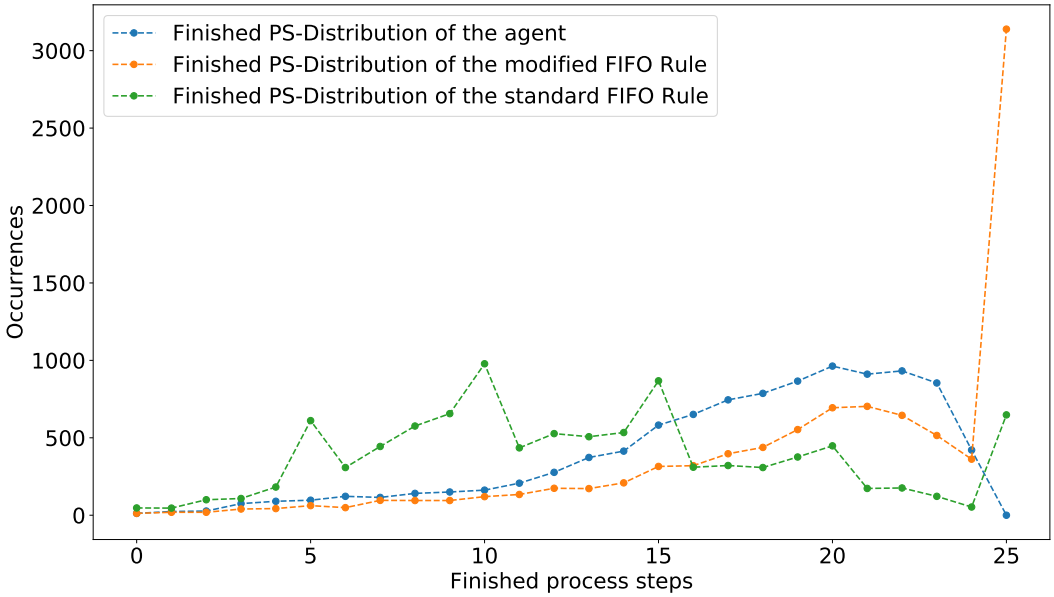


Figure 4.18: Distribution of finished process steps in the last 10000 episodes in comparison to FIFO

4.12 Discussion of the Results

The results show that it is possible to successfully develop an RL-algorithm capable of performing online-reactions in a cell-based manufacturing environment. The algorithm learns to mitigate the negative effects of machine breakdowns and shows better performance than a FIFO scheduling rule.

In the following, the results and the approach are critically examined. The chosen constraints and assumptions that are implemented in the simulation closely represent a real-world manufacturing environment. For further development and even better representation of a physical manufacturing process, the assumed simplifications can be reduced. For example, the supply of data can be modeled more complex and require multiple pre-processing steps before serving as a source of information for the agent. Further, the simulated manufacturing processes can be extended to work continuously and not producing small batch-sizes of products as assumed in this case.

The limiting factor for a complete transition of the trained agent into the real-world is the implemented simulation and not the developed method. The presented method is applicable to any simulation. For a successful transfer of a trained agent into a real-world environment,

it is necessary to develop a more sophisticated simulation that includes all physical aspects of manufacturing.

The online-reaction capability of the agent is strongly dependent on the used hardware and implementation. RL-algorithms are not performing time-intensive calculations and are therefore a valid option for online-reactions in the area of scheduling. The analyzed response time of 0.001 seconds surpasses the required time constraints for online-reactions in the manufacturing process. The response time of the agent can be decreased further by using faster processors and optimized algorithms like parallel computing.

Depending on the NN's design, including connection types, amount of layers and activation functions, the success of the trained agent can be limited. It is not analyzed in this thesis how these elements have to be constructed in relation to each other to achieve even better results. By performing a hyper-parameter grid search, a combination of parameters can be found that will lead to a more adapted agent that can further increase the production rate by maximizing resource utilization.

To evaluate the agent's actual performance, it is necessary to independently analyze the environment in every episode. By only comparing the agent's average performance, the highest possible reward in each episode is neglected. The average reward is strongly impacted by situations where it is not possible to achieve high rewards. These situations occur if all necessary machines for one PS are nonoperational. By comparing the absolute average reward to the baseline, the agent's performance is not viewed relative to the environment and thus lacks informative value. A more informing comparison would be the percentage ratio of the received reward and the highest possible reward in every episode. This comparison is only possible with the use of an improved simulation and independent from the used RL-approach.

The result of this analysis is that it requires a highly developed simulation that incorporates all elements of a physical manufacturing process as well as a tuned set of hyper-parameters to maximize the agent's performance. The presented method is not the limiting factor for a transition into the real-world manufacturing environment.

5 Conclusion

The following chapter summarizes the results of this thesis. In addition, a brief outlook on future possibilities and application of RL in the manufacturing process is presented.

5.1 Summary of Results

This thesis's primary goal is to provide a valid proof of concept for process optimization in the case of machine breakdowns in a cell-based manufacturing environment. The chosen approach to achieve this goal is successfully realized with the help of RL.

The first step in achieving this goal is the identification of the data and its required processing. This goal is achieved, as can be seen by the conducted tests. The agent is capable of understanding the current situation and calculate optimal decisions that significantly increase the production rate. Without the correct section of data and implementation of the state-vector that is presented in this thesis, these results are not possible.

The second step in achieving this goal is to extend the system's possibility to react to changing situations on the shop-floor level without time delays and as close to real-time as possible. This goal is accomplished. The implemented RL-algorithm is able to react to input instantaneously. The average time it took the agent to calculate an action for any given situation is 0.001 seconds. This time is a very reasonable result in the area of online-scheduling as all process steps are not scheduled by milliseconds.

The main objective was to reduce the necessary time it takes to perform all process steps on all products in a dynamic environment with machine breakdowns with the help of suitable reroutings of products. The results show that the agent is capable of performing better than a FIFO scheduling rule. It takes the agent only 30000 episodes to learn the dynamics of the simulation. After that, the agent is continuously performing better than FIFO. This is a clear

sign that the agent understands the correlation between machine breakdowns, its actions and received rewards.

All results indicate that the application of a NN with the combination of an RL-algorithm satisfies all the requirements for a production-management system in the area of a cell-based manufacturing process.

5.2 Outlook

This Bachelor's thesis focuses on providing proof of the concept of the online-scheduling capabilities of RL. Based on the results, further research should be conducted, taking more elements from the real world into account. Data availability or failure probability and the assumed simplification are factors that must be analyzed in detail in physical manufacturing environments. Lager scale tests are required to research further into the capability of RL to control full-scale manufacturing processes.

The cross-influence of different combinations of constants like, for example, learning rate and batch-size that might lead to better results, must be investigated in detail. Especially the design of specialized NNs with adapted activation functions can bear further improvement.

This thesis assumes that the necessary data for constructing the state-vector is always available. This area needs more research as the construction of a state-vector is the main difficulty in today's manufacturing systems. In manufacturing processes with low sensorization, data is not instantly available or not extractable at all. In this case, a promising area of application are algorithms that can work with partially observable environments and directly refer to past events to calculate the best possible action.

The most optimal implementation of agents has to be analyzed to find the best possible solution. Especially the applications and potential of multi-agent-systems have to be inspected. Interconnected and independent decision-making-entities can allow for more optimization and increased capabilities.

With the increasing power of computers and more efficient mathematical functions, more extended and more efficient training of the NNs will be possible in the future. RL is a promising concept that needs to be developed further to exploit all possibilities in the manufacturing environment.

Bibliography

[1] V. Mařík, A. Schirrmann, D. Trentesaux, and P. Vrba, *Software Engineering Methods for Intelligent Manufacturing Systems: A Comparative Survey*. Cham: Springer International Publishing, 2015, vol. 9266, ISBN: 978-3-319-22866-2. DOI: 10.1007/978-3-319-22867-9.

[2] X. Ou, Q. Chang, J. Arinez, and J. Zou, "Gantry work cell scheduling through reinforcement learning with knowledge-guided reward setting," *IEEE Access*, vol. 6, pp. 14 699–14 709, 2018. DOI: 10.1109/ACCESS.2018.2800641.

[3] P. KOPACEK, "Intelligent manufacturing: Present state and future trends," *Journal of Intelligent and Robotic Systems*, vol. 1999, pp. 217–229,

[4] VDMA, "Quick guidemachine learning in mechanical and plant engineering," *VDMA Software and Digitalization*, vol. 2018, pp. 1–34,

[5] Dr. Dirk Hecker, Inga Döbel, Ulrike Petersen, André Rauschert, Velina Schmitz, Dr. Angelika Voss, "Zukunftsmarkt kÜnstliche intelligenz: Potenziale und anwendungen," *Frauenhofer-Allianz Big Data*, 2017.

[6] R. M. Palhares, Y. Yuan, and Q. Wang, "Artificial intelligence in industrial systems," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 12, pp. 9636–9640, 2019, ISSN: 0278-0046. DOI: 10.1109/TIE.2019.2916709.

[7] Y.-C. Wang and J. M. Usher, "A reinforcement learning approach for developing routing policies in multi-agent production scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 33, no. 3-4, pp. 323–333, 2007, ISSN: 0268-3768. DOI: 10.1007/s00170-006-0465-y.

[8] P. Burggraf, J. Wagner, and B. Koke, "Artificial intelligence in production management: A review of the current state of affairs and research trends in academia," in *2018 International Conference on Information Management and Processing (ICIMP)*, IEEE, 12.01.2018 - 14.01.2018, pp. 82–88, ISBN: 978-1-5386-3656-5. DOI: 10.1109/ICIMP1.2018.8325846.

[9] J. P. U. Cadavid, S. Lamouri, B. Grabot, and A. Fortin, "Machine learning in production planning and control: A review of empirical literature," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 385–390, 2019, ISSN: 24058963. DOI: 10.1016/j.ifacol.2019.11.155.

[10] M. Zhao, X. Li, L. Gao, L. Wang, and M. Xiao, "An improved q-learning based rescheduling method for flexible job-shops with machine failures," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, IEEE, 22.08.2019 - 26.08.2019, pp. 331–337, ISBN: 978-1-7281-0356-3. DOI: 10.1109/COASE.2019.8843100.

[11] Tamaki, H., Kryssanov, V.V., and Kitamura, S., "A simulation engine to support production scheduling using genetics-based machine learning," *K. Mertins, O. Krause, and B. Schallock (eds), Global Production ManagementKluwer*, pp. 482–489, 1999.

[12] LUIS M. CAMARINHA-MATOS , HAMIDEH AFSARMANESH, "Editorial: Intelligent manufacturing systems," 1999.

[13] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek, "Optimization of global production scheduling with deep reinforcement learning," *51st CIRP Conference on Manufacturing Systems*, pp. 1264–1269, 2018. DOI: 10.1016/j.procir.2018.03.212.

[14] Dr. S. Archana Bai, "Artificial intelligence technologies in business and engineering," vol. 2011, pp. 856–859,

[15] D. T. Pham and A. A. Afify, "Machine-learning techniques and their applications in manufacturing," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 219, no. 5, pp. 395–412, 2005, ISSN: 0954-4054. DOI: 10.1243/095440505X32274.

[16] J. W. Kim and S. K. Kim, "Interactive job sequencing system for small make-to-order manufacturers under smart manufacturing environment," *Peer-to-Peer Networking and Applications*, vol. 13, no. 2, pp. 524–531, 2020, ISSN: 1936-6442. DOI: 10.1007/s12083-019-00808-1.

[17] A. Goryachev, S. Kozhevnikov, E. Kolbova, O. Kuznetsov, E. Simonova, P. Skobelev, A. Tsarev, and Y. Shepilov, ""'smart factory": Intelligent system for workshop resource allocation, scheduling, optimization and controlling in real time," *Advanced Materials Research*, vol. 630, pp. 508–513, 2012. DOI: 10.4028/www.scientific.net/AMR.630.508.

[18] P. S. Ouelhadj D., "Survey of dynamic scheduling in manufacturing systems," *J Sched (2009)*, vol. 2008, pp. 417–431,

[19] F. Schwartz and S. Voß, "Störungsmanagement in der produktion — simulationsstudien für ein hybrides fließfertigungssystem," *Zeitschrift für Planung & Unternehmenssteuerung*, vol. 15, no. 4, pp. 427–447, 2004, ISSN: 1613-9267. DOI: 10.1007/BF03401248.

[20] Khayal, Osama Mohammed Elmardi Suleiman, *A review for dynamic scheduling in manufacturing*, 2018. DOI: 10.13140/RG.2.2.15345.33129.

[21] W. Chen, J. Li, and W. Ma, "Hybrid flow shop rescheduling algorithm for perishable products subject to a due date with random invalidity to the operational unit," *The International Journal of Advanced Manufacturing Technology*, vol. 93, no. 1-4, pp. 225–239, 2017, ISSN: 0268-3768. DOI: 10.1007/s00170-016-8859-y.

[22] A. Kuhnle and G. Lanza, "Application of reinforcement learning in production planning and control of cyber physical production systems," in *Machine Learning for Cyber Physical Systems*, ser. Technologien für die intelligente Automation, J. Beyerer, C. Kühnert, and O. Niggemann, Eds., vol. 9, Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 123–132, ISBN: 978-3-662-58484-2. DOI: 10.1007/978-3-662-58485-9{\textunderscore}14.

[23] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks - HotNets '16*, B. Ford, A. C. Snoeren, and E. Zegura, Eds., New York, New York, USA: ACM Press, 2016, pp. 50–56, ISBN: 9781450346610. DOI: 10.1145/3005745.3005750.

[24] Weiming Shen, Douglas H. Norrie, "Agent-based systems for intelligent manufacturing a state-of-the-art survey," *Knowledge and Information Systems*, pp. 129–156, 1998.

[25] F. E. Minguillon and G. Lanza, "Coupling of centralized and decentralized scheduling for robust production in agile production systems," *Procedia CIRP*, vol. 79, pp. 385–390, 2019, ISSN: 22128271. DOI: 10.1016/j.procir.2019.02.099.

[26] Dávid Gyulai, András Pfeiffer, Gábor Nick, Viola Gallina, Wilfried Sihn, László Monostori, "Lead time prediction in a flow shop environment with analytical and machine learning approaches," *International Federation of Automatic Control*, vol. 2018, pp. 1029–1034, 2018.

[27] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, "Machine learning in manufacturing: Advantages, challenges, and applications," *Production & Manufacturing Research*, vol. 4, no. 1, pp. 23–45, 2016. DOI: 10.1080/21693277.2016.1192517.

[28] M. Lubosch, M. Kunath, and H. Winkler, "Industrial scheduling with monte carlo tree search and machine learning," *Procedia CIRP*, vol. 72, pp. 1283–1287, 2018, ISSN: 22128271. DOI: 10.1016/j.procir.2018.03.171.

[29] Florian Jaensch, Akos Csiszar, Annika Kienzlen and Alexander Verl, "Reinforcement learning of material flow control logic using hardware-in-the-loop simulation," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 2018. DOI: 10.1613/jair.301.

[30] Karl Kempf, Bruce Russell, Sanjiv Sidhu, and Stu Barrett, "Ai-based schedulers in manufacturing practice: Report of a panel discussion," *AI Magazine Volume 11 Number 5*, vol. 1990, pp. 46–55, 1990.

[31] C. M. Schlick, R. Bruder, and H. Luczak, *Arbeitswissenschaft*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ISBN: 978-3-540-78332-9. DOI: 10.1007/978-3-540-78333-6.

[32] Dipl.-Ing. Markus Decker, Dipl.-Ing. Lamine Jendoubi, "Einführung in die organisation der produktion," 2006.

[33] G. Schuh and V. Stich, *Produktionsplanung und -steuerung 2*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN: 978-3-642-25426-0. DOI: 10.1007/978-3-642-25427-7.

[34] L. Liu, H.-y. Gu, and Y.-g. Xi, "Robust and stable scheduling of a single machine with random machine breakdowns," *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 7-8, pp. 645–654, 2006, ISSN: 0268-3768. DOI: 10.1007/s00170-005-0237-0.

[35] H. Zongxiang, "Research for production scheduling of discrete manufacturing enterprise based on the theory of constraints," in *2008 International Conference on Information Management, Innovation Management and Industrial Engineering*, IEEE, 19.12.2008 - 21.12.2008, pp. 167–171, ISBN: 978-0-7695-3435-0. DOI: 10.1109/ICIII.2008.292.

[36] D. D. Evangelos Kehris, "Application of reinforcement learning for the generation of anassembly plant entry control policy," *CEUR Workshop Proceedings*, vol. 2007,

[37] M. Xu, "Production scheduling of agile manufacturing based on multi-agents," in *2009 Second International Symposium on Knowledge Acquisition and Modeling*, IEEE, 30.11.2009 - 01.12.2009, pp. 323–325, ISBN: 978-0-7695-3888-4. DOI: 10.1109/KAM.2009.193.

[38] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, 1999, ISSN: 03772217. DOI: 10.1016/S0377-2217(98)00113-1.

[39] H. Lu, L. Li, and X. Li, "Research on collaboration-oriented production management in multi-variety and small-batch environment," in *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, IEEE, 29.05.2012 - 31.05.2012, pp. 2778–2781, ISBN: 978-1-4673-0024-7. DOI: 10.1109/FSKD.2012.6234054.

[40] G. Schuh, "Erp enterprise resource planning," in *CIRP Encyclopedia of Production Engineering*, T. I. A. f. Produ, L. Laperrière, and G. Reinhart, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 1–8, ISBN: 978-3-642-35950-7. DOI: 10.1007/978-3-642-35950-7{\textunderscore}6673-3.

[41] Daryl Powell, Erlend Alfnes, Jan Ola Strandhagen, and Heidi Dreyer, "Ifip aict 384 - erp support for lean production,"

[42] D. Natarajan, *ISO 9001 Quality Management Systems*. Cham: Springer International Publishing, 2017, ISBN: 978-3-319-54382-6. DOI: 10.1007/978-3-319-54383-3.

[43] S. C. L. Koh, A. Gunasekaran, and S. M. Saad, "Tackling uncertainty in erp-controlled manufacturing environment: A knowledge management approach," *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 7-8, pp. 833–840, 2006, ISSN: 0268-3768. DOI: 10.1007/s00170-005-0251-2.

[44] Gyanendra Singh, Ajitanshu Mishra, Dheeraj Sagar, "An overview of artificial intelligence,"

[45] R. Fernandes de Mello and M. Antonelli Ponti, *Machine Learning*. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-94988-8. DOI: 10.1007/978-3-319-94989-5.

[46] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, Second edition, ser. Adaptive computation and machine learning series. Cambridge Massachusetts: The MIT Press, 2018, ISBN: 9780262039246.

[47] V. Nasteski, "An overview of the supervised machine learning methods," *HORIZONS.B*, vol. 4, pp. 51–62, 2017, ISSN: 18578578. DOI: 10.20544/HORIZONS.B.04.1.17.P05.

[48] P. Wilde, *Neural Networks Models: An analysis*, ser. Lecture Notes in Control and Information Sciences. Berlin and Heidelberg: Springer, 1996, vol. 210, ISBN: 9783540708193. DOI: 10.1007/BFb0034478.

[49] K. Gurney, *An introduction to neural networks*. London: UCL Press, 1997, ISBN: 1857286731.

[50] Jörg Krüger, Jürgen Fleischer, Jörg Franke, Peter Groche, "Ki in der produktion: Künstliche intelligenz erschliessen für unternehmen," pp. 1–28, 2018.

[51] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, *Activation functions: Comparison of trends in practice and research for deep learning*. [Online]. Available: http://arxiv.org/pdf/1811.03378v1.

[52] S. Urban, "Neural network architectures and activation functions: A gaussian process approach," *Fakultät für Informatik (TUM)*,

[53] A. Hammoudeh, *A concise introduction to reinforcement learning*, 2018. DOI: 10.13140/RG.2.2.31027.53285.

[54] Dranidis, Dimitris and Kehris, Evangelos, "A production scheduling strategy for an assembly plant based on reinforcement learning," *Proceedings of the 5th World Multi-Conference on Circuits, Systems, Communications & Computers, Crete*, 2001.

[55] J. Oh and Y. Kim, "Job placement using reinforcement learning in gpu virtualization environment," *Cluster Computing*, 2020, ISSN: 1386-7857. DOI: 10.1007/s10586-019-03044-7.

[56] T. Altenmüller, T. Stüker, B. Waschneck, A. Kuhnle, and G. Lanza, "Reinforcement learning for an intelligent and autonomous production control of complex job-shops under time constraints," *Production Engineering*, vol. 14, no. 3, pp. 319–328, 2020, ISSN: 0944-6524. DOI: 10.1007/s11740-020-00967-8.

[57] B. Grabot, "Objective satisfaction assessment using neural nets for balancing multiple objectives," *International Journal of Production Research*, vol. 36, no. 9, pp. 2377–2395, 1998, ISSN: 0020-7543. DOI: 10.1080/002075498192580.

[58] Dirk Hecker, Inga Döbel, Stefan Rüping, Velina Schmitz und Angi Voss, "Künstliche intelligenz und die potenziale des maschinellen lernens für die industrie," vol. 2017,

[59] W. Wahlster, "Künstliche intelligenz als grundlage autonomer systeme," *Informatik-Spektrum*, vol. 40, no. 5, pp. 409–418, 2017, ISSN: 0170-6012. DOI: 10.1007/s00287-017-1049-y.

[60] L. I. Yohana Arzi, "Neural network-based adaptive production control system for a flexible manufacturing cell under a random environment," *IIE Transactions*, vol. 1999, pp. 217–230, 1998.

[61] A. Mayr, D. Kißkalt, M. Meiners, B. Lutz, F. Schäfer, R. Seidel, A. Selmaier, J. Fuchs, M. Metzner, A. Blank, and J. Franke, "Machine learning in production – potentials, challenges and exemplary applications," *Procedia CIRP*, vol. 86, pp. 49–54, 2019, ISSN: 22128271. DOI: 10.1016/j.procir.2020.01.035.

[62] *Mnist handwritten digit database, yann lecun, corinna cortes and chris burges*, (visited on 25.08.2020). [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[63] R. Battiti, M. Brunato, and F. Mascia, *Reactive Search and Intelligent Optimization*. Boston, MA: Springer US, 2009, vol. 45, ISBN: 978-0-387-09623-0. DOI: 10.1007/978-0-387-09624-7.

[64] T. Silva and A. Azevedo, "Production flow control through the use of reinforcement learning," *Procedia Manufacturing*, vol. 38, pp. 194–202, 2019, ISSN: 23519789. DOI: 10.1016/j.promfg.2020.01.026.

[65] *Training data for ai in car damage detection for insurance claims*, (visited on 08.09.2020). [Online]. Available: https://www.cogitotech.com/use-cases/insurance/.

[66] Nagdev Amruthnath and T. Gupta, *A research study on unsupervised machine learning algorithms for fault detection in predictive maintenance*, 2018. DOI: 10.13140/RG.2.2.28822.24648.

[67] A. Kuhnle, L. Schäfer, N. Stricker, and G. Lanza, "Design, implementation and evaluation of reinforcement learning for an adaptive order dispatching in job shop manufacturing systems," *Procedia CIRP*, vol. 81, pp. 234–239, 2019, ISSN: 22128271. DOI: 10.1016/j.procir.2019.03.041.

[68] T. Gabel, "Adaptive reactive job-shop scheduling with reinforcement learning agents," *International Journal of Information Technology and Intelligent Computing*, vol. 2008, 2008.

[69] Y. Yue, Y. Tang, M. Yin, and M. Zhou, *Discrete action on-policy learning with action-value critic*. [Online]. Available: http://arxiv.org/pdf/2002.03534v2.

[70] H. van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, IEEE, 1.04.2007 - 05.04.2007, pp. 272–279, ISBN: 1-4244-0706-0. DOI: 10.1109/ADPRL.2007.368199.

[71] ——, "Using continuous action spaces to solve discrete problems," in *2009 International Joint Conference on Neural Networks*, IEEE, 14.06.2009 - 19.06.2009, pp. 1149–1156, ISBN: 978-1-4244-3548-7. DOI: 10.1109/IJCNN.2009.5178745.

[72] Xanthopoulos, A. and Koulouriotis, Dimitrios and Gasteratos, Antonios, "A reinforcement learning approach for production control in manufacturing systems," vol. 2008,

[73] R. Liu and J. Zou, *The effects of memory replay in reinforcement learning*. [Online]. Available: http://arxiv.org/pdf/1710.06574v1.

[74] L. Buşoniu, R. Babuška, and B. de Schutter, "Multi-agent reinforcement learning: An overview," in *Innovations in Multi-Agent Systems and Applications - 1*, ser. Studies in Computational Intelligence, J. Kacprzyk, D. Srinivasan, and L. C. Jain, Eds., vol. 310, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 183–221, ISBN: 978-3-642-14434-9. DOI: 10.1007/978-3-642-14435-6{\textunderscore}7.

[75] L. Busoniu, R. Babuska, and B. de Schutter, "Multi-agent reinforcement learning: A survey," in *2006 9th International Conference on Control, Automation, Robotics and Vision*, IEEE, 5.12.2006 - 08.12.2006, pp. 1–6, ISBN: 1-4244-0341-3. DOI: 10.1109/ICARCV.2006.345353.

[76] *Vacuum-as-a-service: Electrolux trials new subscription-based business models - electrolux group*, (visited on 24.09.2020). [Online]. Available: https://www.electroluxgroup.com/en/vacuum-as-a-service-electrolux-trials-new-subscription-based-business-models-29880/.

[77] S. Baer, J. Bakakeu, R. Meyes, and T. Meisen, "Multi-agent reinforcement learning for job shop scheduling in flexible manufacturing systems," in *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*, IEEE, 25.09.2019 - 27.09.2019, pp. 22–25, ISBN: 978-1-7281-4087-2. DOI: 10.1109/AI4I46381.2019.00014.

[78] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan, "Reinforcement learning algorithm for partially observable markov decision problems," [Online]. Available: https://papers.nips.cc/paper/951-reinforcement-learning-algorithm-for-partially-observable-markov-decision-problems.pdf.

[79] Nobuo Suematsu and Akira Hayashi, "A reinforcement learning algorithm in partially observable environments using short-term memory," [Online]. Available: https://papers.nips.cc/paper/1487-a-reinforcement-learning-algorithm-in-partially-observable-environments-using-short-term-memory.pdf.

[80] G. H. Kim and C. Lee, "Genetic reinforcement learning approach to the machine scheduling problem," in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, IEEE, 21-27 May 1995, pp. 196–201, ISBN: 0-7803-1965-6. DOI: 10.1109/ROBOT.1995.525285.

[81] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, *Safe exploration in continuous action spaces*. [Online]. Available: http://arxiv.org/pdf/1801.08757v1.

[82] OpenAI, *Gym: A toolkit for developing and comparing reinforcement learning algorithms*, (visited on 15.10.2020). [Online]. Available: https://gym.openai.com/envs/BipedalWalker-v2/.

[83] C. Allen, K. Asadi, M. Roderick, A.-r. Mohamed, G. Konidaris, and M. Littman, *Mean actor critic*. [Online]. Available: http://arxiv.org/pdf/1709.00503v2.

[84] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM Journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003, ISSN: 0363-0129. DOI: 10.1137/S0363012901385691.

[85] Konda, Vijay and Tsitsiklis, John, "Actor-critic algorithms," *Society for Industrial and Applied Mathematics*, vol. 2001,

[86] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*. [Online]. Available: http://arxiv.org/pdf/1802.09477v3.

[87] N. M. Seel, *Encyclopedia of the Sciences of Learning*. Boston, MA: Springer US, 2012, ISBN: 978-1-4419-1427-9. DOI: 10.1007/978-1-4419-1428-6.

[88] C. Sammut and G. I. Webb, *Encyclopedia of Machine Learning and Data Mining*. Boston, MA: Springer US, 2017, ISBN: 978-1-4899-7685-7. DOI: 10.1007/978-1-4899-7687-1.

[89] S. Zhang and R. S. Sutton, *A deeper look at experience replay*. [Online]. Available: http://arxiv.org/pdf/1712.01275v3.

[90] Chang Xu, Tao Qin, Gang Wang, Tie-Yan Liu, "An actor-critic algorithm for learning rate learning," *ICLR*, 2017.

[91] C. Xu, T. Qin, G. Wang, and T.-Y. Liu, *Reinforcement learning for learning rate control*. [Online]. Available: http://arxiv.org/pdf/1705.11159v1.

[92] *Setting the learning rate of your neural network.* (visited on 11.09.2020). [Online]. Available: https://www.jeremyjordan.me/nn-learning-rate/.

[93] W. Hu, C. J. Li, L. Li, and J.-G. Liu, *On the diffusion approximation of nonconvex stochastic gradient descent*. [Online]. Available: http://arxiv.org/pdf/1705.07562v2.

[94] Matteo Papini, Matteo Pirotta, Marcello Restelli, "Adaptive batch size for safe policy gradients," *31st Conference on Neural Information Processing Systems*, 2017.

[95] S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team, *An empirical model of large-batch training*. [Online]. Available: http://arxiv.org/pdf/1812.06162v1.

[96] AUDI, "Dialoge smart factory," pp. 1–41, 2017. [Online]. Available: https://www.audi-mediacenter.com/de/publikationen/magazine/dialoge-smart-factory-2017-364.

[97] L. Bom, R. Henken, and M. Wiering, "Reinforcement learning to train ms. pac-man using higher-order action-relative inputs," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, IEEE, 16.04.2013 - 19.04.2013, pp. 156–163, ISBN: 978-1-4673-5925-2. DOI: 10.1109/ADPRL.2013.6615002.

[98] *The neural network zoo - the asimov institute*, (visited on 09.09.2020). [Online]. Available: https://www.asimovinstitute.org/neural-network-zoo/.

[99] *Tensorflow for deep learning*, (visited on 09.09.2020). [Online]. Available: https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html.

[100] ashishpatel26, *Tools to design or visualize architecture of neural network*, (visited on 09.09.2020). [Online]. Available: https://github.com/ashishpatel26/Tools-to-Design-or-Visualize-Architecture-of-Neural-Network.

[101] S. SHARMA, *Activation functions in neural networks*, (visited on 09.09.2020). [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[102] Prateek, *Statistics is freaking hard: Wtf is activation function*, (visited on 18.09.2020). [Online]. Available: https://towardsdatascience.com/statistics-is-freaking-hard-wtf-is-activation-function-df8342cdf292.

[103] V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, *Artificial Neural Networks and Machine Learning – ICANN 2018*. Cham: Springer International Publishing, 2018, vol. 11139, ISBN: 978-3-030-01417-9. DOI: 10.1007/978-3-030-01418-6.

[104] F. Pardo, A. Tavakoli, V. Levdik, and P. Kormushev, "Time limits in reinforcement learning," 2018. [Online]. Available: http://arxiv.org/pdf/1712.00378v3.

[105] Ala'eddin Masadeh, Zhengdao Wang, Ahmed E. Kamal, "Convergence-based exploration algorithm for reinforcement learning," *Iowa State University (ISU), Ames, Iowa, United States*, 2018. [Online]. Available: https://lib.dr.iastate.edu/ece_reports/1/?utm_source=lib.dr.iastate.edu%2Fece_reports%2F1&utm_medium=PDF&utm_campaign=PDFCoverPages.

[106] P. Shyam, W. Jaśkowski, and F. Gomez, *Model-based active exploration*. [Online]. Available: http://arxiv.org/pdf/1810.12162v5.

[107] *Tensorflow*, (visited on 02.09.2020). [Online]. Available: https://www.tensorflow.org/.

[108] *Pytorch*, (visited on 12.10.2020). [Online]. Available: https://pytorch.org/.

[109] K. Team, *Keras: The python deep learning api*, (visited on 02.10.2020). [Online]. Available: https://keras.io/.

[110] W. Fedus, C. Gelada, Y. Bengio, M. G. Bellemare, and H. Larochelle, *Hyperbolic discounting and learning over multiple horizons*. [Online]. Available: http://arxiv.org/pdf/1902.06865v3.

[111] GitHub, *Nikhilbarhate99 - overview*, (visited on 01.10.2020). [Online]. Available: https://github.com/nikhilbarhate99.

Declaration

The submitted thesis was supervised by Prof. Dr.-Ing. Birgit Vogel-Heuser.

Affirmation
Hereby, I affirm that I am the sole author of this thesis. To the best of my knowledge, I affirm that this thesis does not infringe upon anyone's copyright nor violate any proprietary rights. I affirm that any ideas, techniques, quotations, or any other material, are in accordance with standard referencing practices.

Moreover, I affirm that, so far, the thesis is not forwarded to a third party nor is it published. I obeyed all study regulations of the Technical University of Munich.

Remarks about the internet
Throughout the work, the internet was used for research and verification. Many of the keywords provided herein, references and other information can be verifed on the internet. However, no sources are given, because all statements made in this work are fully covered by the cited literature sources.

Munich, November 4, 2020

Jan Nalivaika