Using Deep Neural Networks for Scheduling Resource-Constrained Activity Sequences

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Maschinenbau des
Karlsruher Instituts für Technologie (KIT)
genehmigte

Dissertation

von

M.Sc. Paolo Pagani

Tag der mündlichen Prüfung: 20.07.2022
Hauptreferent: Prof. Dr.-Ing. Kai Furmans
Korreferent: Prof. Dr. Rainer Kolisch

Acknowledgements

This work has been developed during my activity as a research associate at the Institute of Material Handling and Logistics of the Karlsruhe Institute of Technology. I would like to express my gratitude to all people that contributed directly and indirectly to this work.

First of all, I would like to thank my supervisor Prof. Dr.-Ing Kai Furmans for his patient guidance, enthusiastic encouragement and useful critiques on my research work. From the beginning, he provided me with the possibility to create my own research topic and he helped me to find my way both from the research and from the personal point of view. In the last year, he also helped me to find the right motivation and energy to summarize and structure my work of 5 years in this PhD thesis. My grateful thanks are also extended to Prof. Dr. Rainer Kolisch for having accepted to be the second examiner.

Secondly, I would like to thank also all my colleagues for having created a working environment, which has been always stimulating and fun at the same time. They helped me a lot with their constructive feedback about my research topics in many situations and we have spent very nice moments together. A special thank goes to the colleagues of my department, who supported me at most.

Finally, I wish to thank my family and, in particular, my wife Nadja for having been a lovely and perfect wife throughout all these years at the IFL institute giving me always the personal support and the energies to perform at best in my work. My parents and my sister's family also deserve big gratitude for the education and support they have always given me.

Karlsruhe, July 2021 Paolo Pagani

Kurzfassung

Eines der bekanntesten Planungsprobleme stellt die Planung von Aktivitäten unter Berücksichtigung von Reihenfolgenbeziehungen zwischen diesen Aktivitäten sowie Ressourcenbeschränkungen dar. In der Literatur ist dieses Planungsproblem als das ressourcenbeschränkte Projektplanungsproblem bekannt und wird im Englischen als Resource-Constrained Project Scheduling Problem oder kurz RCPSP bezeichnet. Das Ziel dieses Problems besteht darin, die Bearbeitungszeit einer Aktivitätsfolge zu minimieren, indem festgelegt wird, wann jede einzelne Aktivität beginnen soll, ohne dass die Ressourcenbeschränkungen überschritten werden. Wenn die Bearbeitungsdauern der Aktivitäten bekannt und deterministisch sind, können die Startzeiten der Aktivitäten à priori definiert werden, ohne dass die Gefahr besteht, dass der Zeitplan unausführbar wird. Da jedoch die Bearbeitungsdauern der Aktivitäten häufig nicht deterministisch sind, sondern auf Schätzungen von Expertengruppen oder historischen Daten basieren, können die realen Bearbeitungsdauern von den geschätzten abweichen. In diesem Fall ist eine reaktive Planungsstrategie zu bevorzugen. Solch eine reaktive Strategie legt die Startzeiten der einzelnen Aktivitäten nicht zu Beginn des Projektes fest, sondern erst unmittelbar an jedem Entscheidungspunkt im Projekt, also zu Beginn des Projektes und immer dann wenn eine oder mehrere Aktivitäten abgeschlossen und die beanspruchten Ressourcen frei werden.

In dieser Arbeit wird eine neue reaktive Planungsstrategie für das ressourcenbeschränkte Projektplanungsproblem vorgestellt. Im Gegensatz zu anderen Literaturbeiträgen, in denen exakte, heuristische und meta-heuristische Methoden zur Anwendung kommen, basiert der in dieser Arbeit aufgestellte Lösungsansatz auf künstlichen neuronalen Netzen und maschinellem Lernen. Die neuronalen Netze verarbeiten die Informationen, die den aktuellen Zustand der Aktivitätsfolge beschreiben, und erzeugen daraus Prioritätswerte für die Aktivitäten, die im aktuellen Entscheidungspunkt gestartet werden können. Das maschinelle Lernen und insbesondere das überwachte Lernen werden für das

Trainieren der neuronalen Netze mit beispielhaften Trainingsdaten angewendet, wobei die Trainingsdaten mit Hilfe einer Simulation erzeugt wurden.

Sechs verschiedene neuronale Netzwerkstrukturen werden in dieser Arbeit betrachtet. Diese Strukturen unterscheiden sich sowohl in der ihnen zur Verfügung gestellten Eingabeinformation als auch der Art des neuronalen Netzes, das diese Information verarbeitet. Es werden drei Arten von neuronalen Netzen betrachtet. Diese sind neuronale Netze mit vollständig verbundenen Schichten, 1-dimensionale faltende neuronale Netze und 2-dimensionale neuronale faltende Netze. Darüber hinaus werden innerhalb jeder einzelnen Netzwerkstruktur verschiedene Hyperparameter, z.B. die Lernrate, Anzahl der Lernepochen, Anzahl an Schichten und Anzahl an Neuronen per Schicht, mittels einer Bayesischen Optimierung abgestimmt. Während des Abstimmens der Hyperparameter wurden außerdem Bereiche für die Hyperparameter identifiziert, die zur Verbesserung der Leistungen genutzt werden sollten.

Das am besten trainierte Netzwerk wird dann für den Vergleich mit anderen vierunddreißig reaktiven heuristischen Methoden herangezogen. Die Ergebnisse dieses Vergleichs zeigen, dass der in dieser Arbeit vorgeschlagene Ansatz in Bezug auf die Minimierung der Gesamtdauer der Aktivitätsfolge die meisten Heuristiken übertrifft. Lediglich 3 Heuristiken erzielen kürzere Gesamtdauern als der Ansatz dieser Arbeit, jedoch sind deren Rechenzeiten um viele Größenordnungen länger.

Eine Annahme in dieser Arbeit besteht darin, dass während der Ausführung der Aktivitäten Abweichungen bei den Aktivitätsdauern auftreten können, obwohl die Aktivitätsdauern generell als deterministisch modelliert werden. Folglich wird eine Sensitivitätsanalyse durchgeführt, um zu prüfen, ob die vorgeschlagene reaktive Planungsstrategie auch dann kompetitiv bleibt, wenn die Aktivitätsdauern von den angenommenen Werten abweichen.

Abstract

One of the most popular scheduling problems is the scheduling of the activities under precedence and resource constraints. In the literature, this problem is called Resource-Constrained Project Scheduling Problem, or shortly RCPSP, and its goal is generally to minimize the execution time of the entire activity sequence by defining when each activity should start without exceeding the resource consumption. If the activity durations are known and deterministic, it is possible to define their start time à priori with no risk that the schedule becomes unfeasible. However, since the activity durations are often not deterministic and roughly estimated by groups of experts and with historical data, the real unknown activity durations may be different than the assumed ones. In this case, it is generally preferred to use a reactive scheduling policy, i.e. a policy that does not determine the starting time of each activity at the beginning of the project but only determines step by step which activities should be immediately started at each decision point, i.e. at the beginning and every time one or more activities are completed and new resources are released.

In this thesis, a new reactive policy for the Resource-Constrained Project Scheduling Problem is proposed. Unlike the other literature contributions, where exact, heuristic and meta-heuristic methods are used, the proposed approach is based on artificial neural networks and machine learning. The neural networks are used to process the information defining the current state of the activity sequence and to generate priority values for the activities that could be started at the current decision point. On the other hand, machine learning and, in particular, supervised learning are used to train the neural network with training data generated by simulation experiments.

Six different neural network structures are considered in this thesis. They differ both in the input information and in the type of neural network they use to process it. Three neural network types are considered, namely fully connected, 1-dimensional convolutional and 2-dimensional convolutional neural networks. Moreover, within the same structure, different hyperparameters, for example

learning rate, number of epochs, number of layers and number of neurons per layer, are considered and tuned using a Bayesian optimization. The tuning process has also identified which ranges for the hyperparameter should be used to improve the performance.

The best trained neural network is then considered for the comparison with other thirty-four reactive heuristic methods. The results show that the proposed approach is very competitive for what concerns the minimization of the total duration since it outperforms most of the heuristics. Only three heuristics achieved a better performance. However, they require computing times that are many orders of magnitude bigger.

One assumption of this thesis is that, although the activity durations are modeled as deterministic, some deviations may occur during the activity execution. As a result, a sensitivity analysis is performed to test whether the proposed reactive policy remains competitive even if the durations deviate from the assumed value.

Contents

1 Introduction

Isn't it remarkable that of all the
machines devised by the humans,
not one can replace imagination?
-A. Naskar

In the last decades the globalization has increased the competition among companies, which are now concurring on a more and more global market. As a result, they have been facing the need to both improve their productivity and optimize their costs to survive in this highly dynamic environment. One way to do that is to improve the decision-making processes (planning, scheduling, etc.) in production, logistics, project management and so on. If properly done, this can optimize the resource utilization, lower the time required to accomplish tasks, minimize risks and, at the end, lower the overall expected costs.

One of the most popular scheduling problem in companies and in everyday life is the scheduling of activity sequences under resource constraints. For instance, it can be found in a variety of applications such as production planning, project management, maintenance planning, computer resource management and so on. This problem is often modeled as shown in Figure 1.1. Each circle represents an activity and each arrow represents a precedence relation, namely that the activity after the arrow cannot start if the activity before the arrow is not completed. Moreover, every activity has a deterministic duration and a list of required resources which are available in a limited quantity and shared among all the activities. This problem is commonly called Resource-Constrained Project Scheduling Problem (RCPSP), and the goal is to define scheduling policies that minimize the project makespan, i.e. the time required to complete all the activities (Kolisch and Padman (2001)). As Moehring, Schulz, Stork, and Uetz (2003) stated, due to its universality, this problem is one of the most intractable problems in operations research and, therefore, it became a popular playground for the latest optimization techniques, including virtually all local

search paradigms. Moreover, the solution of the problem is NP-hard (Blazewicz, Lenstra, and Kan (1983)) according to the computational complexity theory, which means that it cannot be solved in polynomial time.

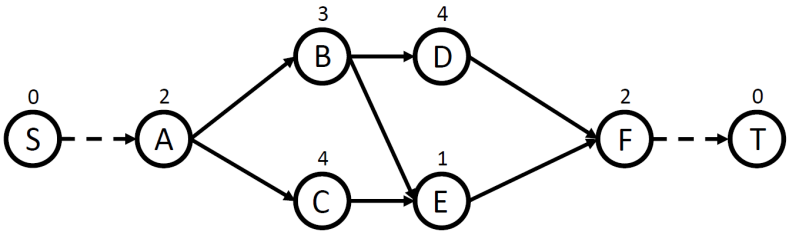

Figure 1.1: Example of activity sequence represented according to activity-on-the-node convention (Davis (1973)).

Apart from the standard version of this problem, many variations can be found in the literature. In fact, a lot of effort has been done to study the so-called multi-mode RCPSP (Sabzehparvar and Seyed-Hosseini (2008), Van Peteghem and Vanhoucke (2011)), i.e. a resource-constrained project scheduling problem where the single activities can be executed with different durations and/or a different number of required resources. Another variation is the so-called multi-skill RCPSP (Neron (2002), H. Zheng, L. Wang, and X. Zheng (2017)), i.e. a resource-constrained project scheduling problem where the resources are able to perform different functions and every activity requires a certain amount of each function to be executed. Moreover, in the literature there are also works where the objective function has been varied from the original version of the problem. For example, Chand, Singh, and Ray (2016) proposed a solution for the RCPSP that aims to achieve the best compromise between make-span and robustness, which is the probability that the initial schedule (i.e. the starting times of the activities) remains feasible through the project execution.

In real-world applications, the execution times of an activity can be rarely considered to be deterministic. Also in the context of the RCPSP, many literature contributions have considered stochastic durations in the so-called Stochastic-RCPSP (S-RCPSP). However, the expected value and distribution of the activity durations are generally estimated either by a group of experts or using historical data. In both cases, the estimation implies some deviations between the assumed probability distributions and the real ones and an à priori schedule, which implies the definition of the starting time of all activities prior to the project's begin, may become unfeasible. As a consequence, the scheduler

must either define some time buffers between two consecutively scheduled activities (Van de Vonder, Demeulemeester, Herroelen, and Leus (2005)) or define a reactive scheduling policy to plan step by step (Rostami, Creemers, and Leus (2015)). This second approach is considered in this thesis and consists in defining which activities should be immediately scheduled at each decision point, i.e. at the project's beginning and whenever another activity has been just completed and new resources have become available.

In the literature, the following 3 classes of solutions have been applied to the RCPSP:

- **Exact methods** (Patterson (1984)): Exact methods can only be computed for a limited number of activities, around 30 and 50 according to Abdolshah (2014) and with deterministic durations, since the computing time rapidly increases with the number of considered activities in the sequence and with its complexity (e.g. number of resource types).
- **Heuristics** (Kolisch and Hartmann (2006)): Heuristics generally provide good solutions within a reasonable computing time and they are applicable to much larger and more complex activity sequences and, most of them, also with stochastic activity durations.
- **Meta-Heuristics** (Myszkowski, Skowronski, Olech, and Oslizlo (2015), Jia and Seo (2013), Zamani (2013), Agarwal, Colak, Deane, and Rakes (2014)): Meta-heuristics are general algorithmic frameworks, often nature inspired, designed to solve complex optimization algorithms. They generally provide better solutions than the heuristics, but they generally require more computing time.

Although the literature on the RCPSP is very rich, no contributions could be found where the RCPSP was tackled with machine learning-based algorithms and, in particular, using deep artificial neural networks. Mao, Alizadeh, Menache, and Kandula (2016) has already tested a similar approach in a similar problem with promising results. In particular, artificial neural network are characterized by a big training time but a low decision time, since only simple and highly parallelizable operations are required once the neural network has been trained.

The goal of this thesis is to close this research gap by investigating the use of deep neural networks as reactive scheduling policy for the deterministic resource-constrained project scheduling problem (S-RCPSP) in its original formulation.

This approach presents a great opportunity to design competitive, fast and generalized decision tools for a step by step scheduling. One of the main advantages of this solution in comparison to other state-of-the-art algorithms is its capacity to schedule completely new activity sequences without running a new optimization, which implies faster decisions. Although the activity durations are modeled as deterministic, it has been chosen to consider only reactive scheduling policies and to assess in the last part of the thesis what happens if the durations are affected by a certain level of uncertainty with a sensitivity analysis.

1.1 Problem description

Deep neural networks for decision-making are used more and more often in the recent years. Since every problem has its own peculiarities and the possibility to apply such neural networks to the resource-constrained project scheduling problem has not been investigated yet, many research questions related to this approach must be still answered.

Reactive algorithms can take into consideration the past, present and/or future information in the decision-making process. In the RCPSP the available information includes the already completed activities, the ones in progress and ones to be started in the future. How exactly this information is prepared and processed by the neural network to come out with a decision is still an open issue in the RCPSP and will be tackled in this thesis with the following research questions:

> **First cluster of questions: Deep neural network as a decision tool for reactive decision making process.**
>
> Which input information can be used? How should the input information be prepared before being processed by the neural network? How does the input and output information structure look like? How is the output information used to define which activities should be immediately scheduled?

Once the input and output information structure has been defined, it is required to generate and prepare the data for training of the neural network. This issue will be tackled with the following research questions:

> **Second cluster of questions: Preparation of the data for training.**

How to generate the training data? How to divide the set of activity sequences to assure that the neural network learns in a generalized way? How many projects are required to generate a sufficient number of training data avoiding overfitting?

After the data have been prepared, the training takes place. Even for neural networks applied to other problems in the literature there are no clear rules and guidelines about which training parameters and which type of neural network should be chosen for the training. As a result, these further research questions will be answered in this thesis:

Third cluster of questions: Training.

Which hyperparameters of the neural networks (e.g. learning rate, number of epochs, dropout values, number of layers, number of neurons or filter per layer and so on) should be used? Which neural network type (e.g. fully connected, convolutional, etc.) is the most suitable one?

Once the best neural network topology and the best hyperparameters among the tested ones have been chosen, it is interesting to compare the approach proposed in this thesis with other algorithms in terms of both project makespan minimization and computing time for activity sequences with different sizes.

Fourth cluster of questions: Benchmark after training.

How does the proposed scheduling policy perform in absolute terms? How does it perform in comparison to other algorithms considering both the makespan and the decision time?

The fifth and last cluster of questions verifies whether and how much the performance of the proposed reactive scheduling policy decreases if the decisions are taken using the assumed values for the durations but the real values are different.

Fifth cluster of questions: Sensitivity analysis.

How does the performance of the proposed approach change if the estimated activity durations are affected by uncertainty? How does the proposed approach perform compared to the other considered reactive scheduling policies in these uncertain circumstances?

1.2 Structure of the thesis

Based on the research questions and the problem statement just described, this thesis is structured in seven chapters.

After the introduction of Chapter 1, where the motivation and the scope of the thesis are presented, the Resource-Constrained Project Scheduling Problem is presented in Chapter 2. Since many versions of the problem can be found in the literature, it is explained which version will be considered and a formal definition is given. Moreover, the concept of reactive scheduling policy is introduced. In the final part of the chapter, the performances of different state-of-the-art algorithms are analyzed and the research gaps are identified.

In Chapter 3, some basic knowledge about deep neural networks and the correspondent decision models are provided. First of all, some key concepts like, for example, the training of machine learning models and hyperparameter tuning are introduced. After that, the author addresses specifically the artificial neural networks by introducing the types used in this thesis and by representing how they have been used in previous literature contributions related to different fields as decision models.

The methodology used to apply them to the RCPSP is presented in Chapter 4, where the used input and output information and the neural network structure is presented. In particular, it is explained how the current state of the activity sequence can be converted into an input information of a fixed dimension, how the information is processed by different neural networks to create the output vector and how the latter is used to take decisions about the activities to be immediately scheduled at each decision point. Moreover, the hyperparameters considered in the hyperparameter tuning are presented. Finally, the project generator used to randomly create new project instances is introduced and the considered project class is defined.

Chapter 5 reports the results of both the hyperparameter tuning and the final evaluation on the test projects. The first aims to find out which hyperparameter values are suitable for the scheduling in the given class of projects, while the goal of the second one is to measure the performance of the proposed approach on activity sequences that have not been used during neither the training nor the hyperparameter tuning and to compare the proposed scheduling policy with the considered heuristics.

The fifth cluster of questions is answered in Chapter 6. Using a sensitivity analysis, the performances of the reactive scheduling policies are measured considering different levels of uncertainty for the activity durations. In particular, it is assumed that each policy schedules the activities at each decision point considering the assumed values for the durations but the real ones are drawn from a uniform distribution with expected value equal to the assumed one and a standard deviation defined by different uncertainty levels.

The final chapter summarizes the results and an outlook on further research areas related to the approach presented in this thesis is given.

2 The resource-constrained project scheduling problem (RCPSP)

The world is one big data problem.
-A. McAfee

In this chapter the resource-constrained project scheduling problem (RCPSP), is presented and an overview about the related literature is given. First of all, a definition of the problem is given, its graphical representation is described and concrete examples of real-world applications of the problem are presented. Secondly, the state-of-the-art algorithms investigated in the literature are briefly mentioned.

2.1 Project as an activity sequence

Projects nowadays are omnipresent and are encountered in many fields of life, such as business, leisure or social activities (Schwindt and Zimmermann (2015a)). According to Kerzner (2017) a project is any series of activities and tasks that:

- has a specific objective to be completed within certain specifications
- holds defined start and end dates
- has funding limits (if applicable)
- requires human and nonhuman resources, for example people, money or equipment
- is multifunctional, for example cut across several functional lines

The Project Management Institute (PMI) defines a project as a temporary endeavor undertaken to create a unique product or service (Rose (2013)). Following both definitions, a project can be understood as a one-time endeavor, consisting of a set of activities, whose execution requires time, consumes resources and leads to a cash flow or entails costs (Schwindt and Zimmermann (2015a)). Moreover, precedence relations can exist between individual activities. These precedence relations are defined by technological or organizational requirements that constrain the activities to be executed in a specific order with respect to their timing relative to each other. Thus, a precedence relation establishes a constraint between two activities, not allowing the second activity to start before the first activity has been completed (Schwindt and Zimmermann (2015a)).

2.2 Representation of a project

A network diagram has proven to be a solid method for displaying a wide variety of activity planning and scheduling problems (Davis (1973)). A project whose activities and precedence relations have been identified, can be illustrated as a network diagram consisting of nodes and arcs and can be expressed by a graph, denoted as G(N,A), where N represents the set of nodes and A represents the set of arcs. A set of activities and their precedence relations can be displayed as a network diagram using two formats: an activity-on-the-node (AoN) and an activity-on-the-arc (AoA) (Vanhoucke (2012)). Both forms of representations are introduced in the following paragraphs with Figure 2.1 and 2.2 representing the same activity sequence modeled in the two formats.

2.2.1 Activity-on-the-arc representation

In an AoA diagram, arcs represent activities and nodes are events or milestones which represent the beginning and/or the end of a set of activities. Information about the activities such as unique identifiers and activity durations are contained on the arcs. In an AoA network, dummy activities are required in order to be able to display all precedence relations between activities of a project. Dummy activities are represented by dashed arrows and consume neither time nor resources. While the AoN network always uniquely represents a network, the AoA format cannot guarantee a unique representation of a project due the use of dummy arcs. Figure 2.1 represents the graph for the AoA format.

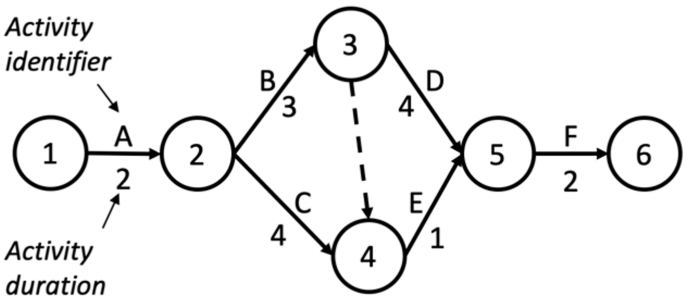

Figure 2.1: Example of project represented according to activity-on-the-arc format.

2.2.2 Activity-on-the-node representation

In an AoN network, the nodes denote activities while the arcs depict precedence relations between activities. Unique identifiers and activity durations of the activities are attached to the nodes. A directed arc between two activities represents a precedence relation which implies, that the activity at the end of the arc cannot be started before the activity at the beginning of the arc has been finished. In the AoN diagram, dummy nodes with durations of zero represent the start and the end of the project. Both dummy nodes are connected to the project activities with dashed arrows. Figure 2.2 represents the graph for this format.

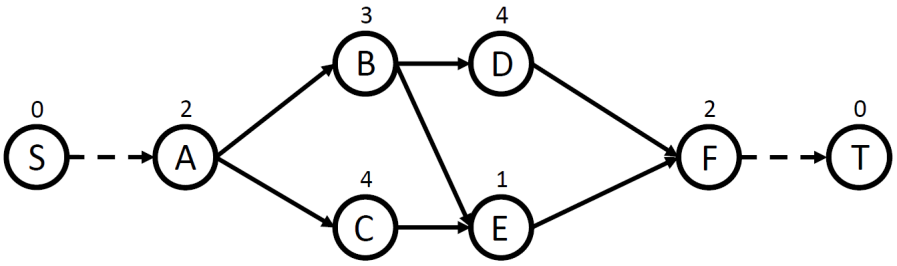

Figure 2.2: Example of project represented according to activity-on-the-node format.

Both formats are legitimate alternatives for displaying a project and contain the same content of information Vanhoucke (2012). However, the AoN network guarantees a unique representation of a project and, as a result, it will be used in this thesis.

2.2.3 Activity representation with resource consumption

This thesis deals with the resource-constrained project scheduling problem (RCPSP) which means that each activity, i.e. each node, has a specific resource consumption. The latter is represented as depicted in Figure 2.3 where a generic activity j requires $r_{j,1}$ pieces of resource type 1, $r_{j,2}$ pieces of resource type 2 and so on until resource type K.

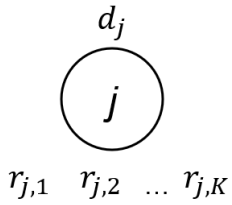

Figure 2.3: Example of activity with resource consumption according to activity-on-the-node format.

2.3 Definition of project management and project life cycle

Project management entails all activities that deal with the initiating, planning, decision making, executing, monitoring and controlling process in the context of a project (Schwindt and Zimmermann (2015a)). Hence, project management requires the application of knowledge, skills, tools and techniques to successfully meet the project's interests and control the achievement of a project's objectives (Munns and Bjeirmi (1996)). Seeing projects from a project management point of view, most projects go through similar phases from their initiation until their completion. The project life cycle usually consists of the project conception, project definition, project planning, project execution and project termination. However, these phases can be carried out differently in different organizations and they also may vary in their size and complexity (Klein (1999)). An illustration of the project life cycle phases is shown in Figure 2.4.

Again according to Klein (1999) the phases can be described as follows: during the project conception phase a vague idea of a project is at hand and economic and risk analyses are performed to prove the feasibility of the project and whether to implement it or not. In the project definition phase the project's

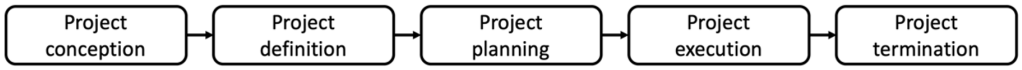

Figure 2.4: Project life cycle phases.

objectives are set and the project's general organization is laid out. The project planning phase begins with breaking down a project's workload into several sub-activities and identifying precedence relations between these sub-activities in order to create a project schedule that allows for processing these sub-activities in a structured manner. While the so far described phases dealt with the preparation of a project for its implementation, in the following project execution phase the project is executed and its progress is being monitored. In the final project termination phase the project is being reviewed whether it has met its objectives and it is evaluated in order to detect potential improvements for future projects.

The given information on the project life cycle phases are meant to give a general overview and illustrate the role of the project scheduling among them. Because of their relevance for this thesis, the next section covers more details on project planning and scheduling.

2.4 Basics of project scheduling

As described in the previous section, during the planning phase of the project life cycle a project is broken down into precedence-related sub-activities. The objective of project scheduling is then to generate a timetable that assigns start and end dates for these sub-activities while maintaining certain scheduling goals, like, for example, minimizing the project duration, also called makespan (Vanhoucke (2012)). Additionally, the project schedule needs to adhere to the precedence relations and the potential further project characteristics like resource constraints or costs (Klein (1999)).

Depending on what kind of constraints are taken into account, the literature distinguishes between time-constrained and resource-constrained project scheduling problems (Schwindt and Zimmermann (2015a)). In time-constrained scheduling problems, precedence relations are the only constraints that need to be considered for the creation of the project's schedule. There are no limitations regarding the availability of resources, hence all required resources for the

sub-activities can be provided in any desired amounts. In resource-constrained scheduling problems, each activity does not only requires some time to be completed but also a set of resources. The resources can be of the same type or they can belong to different resource types. In general, in order to model the resource request as a constraint, the resources must be available in a limited amount and they must be shared among all the activities. In this thesis the resource-constrained scheduling problem will be considered.

2.5 Resource-constrained project scheduling problem in the literature

A huge number of variations of the problem and their corresponding solution methods can be found in the literature. The simplest version is the time-constrained problem, i.e. without considering the resource constraints. This problem has been already formally defined in the 50s with the so-called Critical Path Method (CPM) and the Program Evaluation and Review Technique (PERT). The first one is used when the activity durations are deterministic and the second one when they are stochastic.

Nevertheless, the resource-constrained version of the problem (RCPSP), has been much more investigated in the literature due to fact that optimal solutions for sequences with a number of activities greater than 50 are generally unknown (Abdolshah (2014)). According to Habibi, Barzinpour, and Sadjadi (2018), the different problem sub-classes and the correspondent solution approaches can be summarized in Figure 2.5.

2.5.1 Resources

As mentioned in the previous sections, in the RCPSP limited amounts of resources are available to perform the project activities. Prior to the beginning of the project, all resources are available and are allocated in so-called resource pools (one resource pool for each resource type). Each activity of the project requires a certain amount of resource units and, as soon as it starts, these resources are allocated to it.

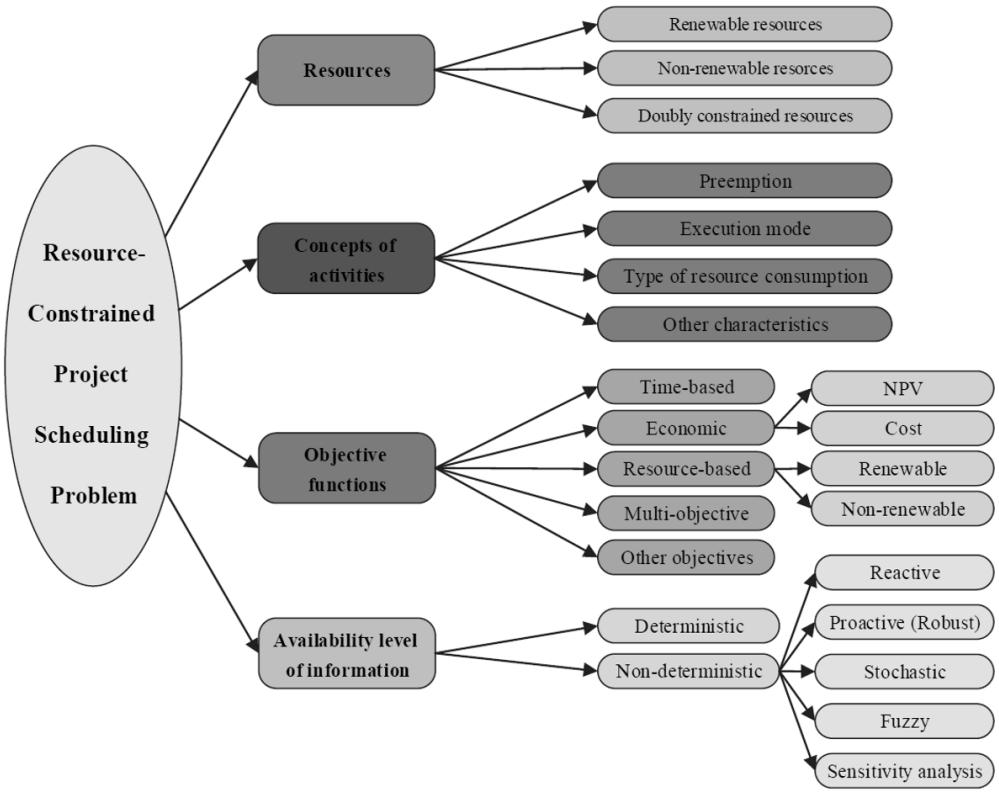

Figure 2.5: Classification of resource-constrained project scheduling problems (Habibi, Barzinpour, and Sadjadi (2018)).

The resource use can be modeled in three different ways:

- **Renewable resources.** If each resource unit can be reused for an unlimited amount of times along the project execution. After the completion of each activity, the allocated resources are released back into the shared resource pools. If an activity cannot start due to the lack of some resources, it must wait until other activities are completed and they release the resources back into the pools. In this regard, human resources and machinery are generally considered to belong to this class of resources (Carlier and Moukrim (2015)).
- **Non-renewable resources.** If a resource unit can be used only once, namely that after the completion of a each activity, the allocated resources are consumed and they are not release back into the pools (Slowinski (1980)). If an activity cannot start due to the lack of some resources, a deadlock occurs and the project cannot be completed. The required raw material (Kyriakidis, Kopanos, and Georgiadis (2012)) and the project

budget (K. Wang, Chi, and Wan (1993)) are two of the most significant examples of non-renewable resources.

- **Doubly constrained resources.** This category of resources has the properties of both renewable and non-renewable resources. In other words, the access to these resources is limited both in each period and the total duration of the project (Kyriakidis, Kopanos, and Georgiadis (2012). Other examples of this type of resources could be energy in a scenario where there is a power limit (energy over time) and at the same time there a limited project energy budget.

For all these three categories, it is generally assumed that the amount of resource consumption and available capacity of each of them are in the form of integers but it is also possible to find some contributions where the resource availability and consumption are (positive) real numbers.

2.5.2 Concepts of activities

A second important feature in characterizing the sub-class of considered RCPSP problem is the characterization of the activities.

- **Preemptive scheduling.** One of the main assumptions related to the basic RCPSP is that the activities of the project cannot be interrupted or discontinued. In other words, an activity must continue till its end without any interruption. However, in practice, it may happen that some activities must be stopped during the processing, which can be due to destruction or disabilities of resources, equipment repair, etc. (Chen and Z. Zhang (2016)). As a result, a new kind of project scheduling arises, namely the Preemption-RCPSP (P-RCPSP). Further variations of the P-RCPSP also exist.
- **Activity execution mode.** In the basic RCPSP, it is assumed that the activities can be performed only in one way, i.e. with a predefined processing time and resource consumption. Elmaghraby (1977) proposed for the first time the assumption of multi-mode activities, in which several techniques or alternatives exist to complete them, each having its own specific duration and resource consumption (Hartmann and Briskorn (2010)). The project manager must not only choose which activities should be started but also with which mode. This type of project scheduling problem is

called the Multimode Resource-Constrained Project Scheduling Problem (M-RCPSP). Further variations of the M-RCPSP also exist.

- **Type of resource consumption.** In the basic RCPSP, it is assumed that the required amount of resources during each activity execution is constant. This assumption can be changed into a more general one by considering a resource consumption changing during the execution (Hartmann and Briskorn (2010)).
- **Other characteristics.** Many other variations of the basic RCPSP concerning the characteristics of the activities can be found in the literature. The complete list can be found in Habibi, Barzinpour, and Sadjadi (2018). The last one which is noteworthy is the Multiskill RCPSP (MS-RCPSP) (Neron (2002), H. Zheng, L. Wang, and X. Zheng (2017)). In this version of the problem, the activities do not require a specific list of resources directly but they require a predefined amount of skills. The available resources provide a certain amount of one or more skills. As a result, the same skill requirement of an activity can be satisfied by different sets of resources. At each decision point, the project manager must decide which activities will be started and which resource units will be allocated to each of the chosen activities.

2.5.3 Objective functions

As mentioned in the previous sections, the goal of the RCPSP is to use a policy to schedule the activities in such a way that the project goal is met. The project goal is generally an objective function that the project manager tries to optimize (minimization or maximization). Various objective functions are used in the literature and they can be classified in the following way:

- **Time-based objective functions.** The minimization of the project completion time is the most popular type of time-based objective function in the RCPSP literature. In this case, the decision maker tries to schedule the activities of the project in a way that the completion time of the project reaches the minimum amount (Creemers (2015)). Another time-based objective function is the tardiness of the activities. In this problem, the project manager aims to minimize the delays of the activity completions from the correspondent due dates.

- **Economic objective functions.** Cost-based (Berthaut, Pellerin, Perrier, and Hajji (2014)) and Net Present Value (Wiesemann, Kuhn, and Rustem (2010)) objective functions are the most used among the economic ones. Both consider that the cash flows (revenues or costs) take place while starting, completing or performing activities and consuming resources and that there may be a trade-off between cost and time. For instance, in the multi-mode project scheduling problems, the activity durations and the costs may vary for each execution mode. Since the completion of an activity in a shorter duration is often associated with increased costs, the decision maker must evaluate if the extra costs are compensated by a significant advantage in terms of project duration. While the cost-based objective function considers the total revenues (or costs) during the project no matter when they occur, the Net Present Value objective function considers that they take place at different points in time and uses the Net Present Value instead to have a better estimate of the financial impact of the decisions during the project execution.
- **Resource-based objective functions.** The Resource-based objective functions can be divided into renewable resource objective functions and non-renewable resource objective functions. To the first group belongs, for example, the minimization of the total costs related to the provision of a specific level of resource capacity without exceeding the project deadline (Moehring (1984)), while the second group deals with the minimization of the consumed non-renewable resources without exceeding the project's due dates.
- **Other objectives objective functions.** For example, Abbasi, Shadrokh, and Arkat (2006) presented a RCPSP model with two objectives. On one hand, the minimization of the project execution time and, on the other hand, the maximization of scheduling robustness, i.e. the probability that the schedule stays feasible even if the activity duration have some deviations.
- **Multi-objective objective functions.** In case more objective functions are considered simultaneously and the decision maker aims to achieve the best compromise by combining them in a unique objective function.

2.5.4 Availability of information

The aim of project scheduling is providing a baseline schedule for a more accurate control of the project and an easier planning for project success. However, it is noteworthy that this schedule is generally defined before the project's beginning and its accuracy depends on the information used to provide the scheduling timetable. The availability level of the required information can be full or limited. If the available information is complete and reliable, the obtained scheduling timetable remains valid for sure. Otherwise, the scheduling process must be done in an uncertain environment (Habibi, Barzinpour, and Sadjadi (2018)) and, if needed, updated during the project execution.

- **Deterministic.** If all the information (e.g. activity durations, resource availability and request and deadlines) is known in advance (Fang, Kolisch, L. Wang, and Mu (2015)).
- **Non-deterministic.** If some pieces of information are missing. According to Habibi, Barzinpour, and Sadjadi (2018), a non-deterministic scheduling can be further classified in the following subclasses:
  - **Reactive scheduling.** This approach is applied during the project execution and is based on the information available at the decision moment. In this case, the decision maker tries to find the optimal schedule by using the assumed deterministic values without taking into account the uncertainties about the project's characteristics. Nevertheless, the chosen schedule is reviewed and re-optimized in case of unexpected deviations, e.g. if an activity takes longer than expected.
  - **Stochastic scheduling.** In this approach, it is assumed that some of the project's parameters are randomly distributed with a known distribution. The literature related to stochastic project scheduling can be categorized into four problems of Stochastic Resource-Constrained Project Scheduling Problem (S-RCPSP) with the same assumptions as the basic RCPSP but with stochastic activity durations, project scheduling problems with stochastic activity interruptions, stochastic discrete time/cost trade-off problems and stochastic project scheduling problems with economic objective functions.

- **Fuzzy scheduling.** In some cases, lack of historical data might lead to inability to determine the probability distribution for duration of activities. In addition, the duration of activities is estimated by experts often under unique conditions of the project. In such conditions, where there is more ambiguity than uncertainty, the use of fuzzy numbers is more preferred for modeling, compared to random variables. In other words, these values use membership functions based on possibility theory as a replacement for probability distributions (Herroelen and Leus (2005)).
- **Proactive (robust) scheduling.** In contrast to the concept of reactive scheduling, proactive scheduling aims to create robust schedules, i.e. schedules that do not require major changes if some unexpected events occur. For instance, it is possible to schedule a time buffer between the expected completion of an activity and the beginning of the following one. In this case, even if some activities have a delay, the completion time of the project may not change (Palacio and Larrea (2017)).
- **Sensitivity analysis.** This approach is not properly a scheduling algorithm but more a verification tool to understand how good a schedule or a scheduling algorithm remains, when some parameters change, like for example the activity durations. The used scheduling policy takes decisions and creates schedules considering the known information about the project, while the real activity durations may be different.

2.5.5 Considered scheduling problems in this thesis

In this thesis two different problems are considered.

The first one will be denoted as deterministic RCPSP in the following chapters and refers to the basic RCPSP, i.e. the Resource-Constrained Project Scheduling Problem with the following characteristics:

- Deterministic activity durations.
- Renewable resources with known total availability.
- No Preemptions.
- Single-mode activities.

- Predefined resource consumption for each activities.
- With the project total duration, or project makespan, as objective function to minimize.

The second one is a sensitivity analysis to test the proposed decision tools when the activity durations deviate from the assumed value. This second problem has the following characteristics:

- Sensitivity analysis assuming random deviations of the activity durations. In particular, the real unknown values are drawn from a uniform distribution with expected values equal to the assumed ones, which are the ones used by the scheduling policy in the scheduling process, and different standard deviations related to 11 levels of uncertainty.
- Renewable resources with known total availability.
- No Preemptions.
- Single-mode activities.
- Predefined resource consumption for each activities.
- With the project total duration, or project makespan, as objective function to minimize.

2.6 Scheduling algorithms for the deterministic RCPSP

In the previous paragraphs the different classes of the RCPSP have been presented without considering the scheduling algorithms in detail. The purpose of this section is to present some methodologies and algorithms used to schedule the activities of the projects. First of all, a formal definition of the problem is given. After that, the state-of-the-art algorithms to solve it are presented. Finally, the literature contributions to the sensitivity analysis for the RCPSP are analyzed.

2.6.1 Formal definition of the Resource-Constrained Project Scheduling Problem

Formally, the deterministic RCPSP can be expressed like a single project consisting of a set of J activities with known deterministic durations $d_j \in \mathbb{N}$ for each

activity $j \in \mathbb{J}$. All activities are executed without preemption, which means that once an activity is started, it is executed without interruption until its completion. The project is finished when all the activities have been completed. Furthermore, $\mathbb{K}$ is a set of renewable resource types and each resource type $k \in \mathbb{K}$ has a finite capacity R_k that remains constant throughout the project execution. Each activity $j \in \mathbb{J}$ requires $r_{j,k}$ units of each resource type $k \in \mathbb{K}$ for the entire duration of its execution. It is assumed that $0 \leq r_{j,k} \leq R_k$. Two additional dummy activities 0 and $J+1$ are considered. They represent the start and the end of the project and require neither time ($d_0 = d_{J+1} = 0$) nor resources ($r_{0,k} = r_{J+1,k} = 0$ for all $k \in K$) to be executed.

A solution to an instance of RCPSP is a schedule, which is denoted by a vector $s = (s_0, ..., s_{J+1})$, in which s_j is the starting time of activity $j \in \mathbb{J}$. Without loss of generality, starting times are restricted to integer values. The starting times must respect a given set of precedence constraints, which are described by a directed an acyclic graph $G(\mathbb{J}, \mathbb{A})$ with $\mathbb{A}$ a set of partial-order relations on $\mathbb{J}$ (a binary relation that is transitive and irreflexive). Such relations $\mathbb{A}$ on $\mathbb{J}$ are called precedence relations. Activity 0 has no predecessors and activity $J+1$ has no successors. The conceptual formulation of the RCPSP is the minimization of s_{J+1}, i.e. the project completion time or project makespan MS_p, under the following constraints:

$$s_j + d_j \leq s_i \qquad \forall (j,i) \in \mathbb{A} \tag{2.1}$$

$$\sum_{j \in \mathbb{J}(s,t)} r_{j,k} \leq R_k \qquad \forall t \in \mathbb{N}_0, \forall k \in \mathbb{K} \tag{2.2}$$

$$s_j \in \mathbb{N} \qquad \forall j \in \mathbb{J} \tag{2.3}$$

The constraint set 2.1 describes the precedence constraints between the activities and are all finish-to-start (FS) constraints, namely that the successor activity cannot be started before the predecessor is completed. The constraint set 2.2 represents the resource constraints, where the set $\mathbb{J}(s,t)$ contains the activities that are in process during time period t (time interval $[t-1, t]$) according to schedule s (see equation 2.4).

$$\mathbb{J}(s,t) = j \in \mathbb{J} : s_j \leq (t-1) \wedge (s_j + d_j) \leq t \tag{2.4}$$

A schedule s that respects the constraints 2.1, 2.2, 2.3 and 2.4 is called a feasible schedule.

2.6.2 Scheduling algorithms

Surveys of solution methods for RCPSP are provided in Demeulemeester and Herroelen (2002) and Neumann, Schwindt, and Zimmermann (2012) and, in general, it is possible to divide the scheduling algorithms for the considered problem in three classes: exact methods, heuristics and meta-heuristics. The three classes are explained in the paragraphs below.

Exact methods

With exact methods it is possible to compute the best schedule (or schedules), i.e. the one with the lowest possible makespan. However, they can compute the optimal solution only for projects with a maximum number of activities around 30 and 50 (Abdolshah (2014)). In fact, their computing time increases exponentially as the number of considered activities in the project and its complexity (e.g. number of resource types or number of precedence constraints) increases.

Exact methods are based on enumeration algorithms, methodically searching the set of possible solutions in such a way that not all possibilities need be considered individually. Exact RCPSP solving methods comprise among others dynamical planning and branch and bound methods Demeulemeester and Herroelen (1992). In Patterson (1984), a set of those methods have been investigated and compared on a set of project instances. The methods differ in how the tree representing partial schedules is generated and saved, and in the methods used to identify and discard inferior partial schedules.

Searching the entire problem space makes the exact methods most of the times inapplicable for practical applications Agarwal, Colak, and Erenguc (2015). The majority of the contributions on exact approaches mainly aim to generate benchmark solutions for small activity sequences to evaluate new approximate methods (Kolisch and Hartmann (1999)).

Heuristics

While various exact methods have been described in the literature for obtaining optimal solutions for the RCPSP, the development of heuristic procedures has also received extensive attention as the computing time required for finding a guaranteed optimal solution becomes unacceptably large as the size and complexity of the instances increases. Heuristics generally provide a good solution within a reasonable computing time and they are applicable to much larger and more complex activity sequences and, most of them, also with known stochastic activity durations or under uncertainty.

Heuristics compute and assign priorities to the activities according to an algorithm. After the priorities have been assigned, it is possible to generate a schedule step by step by scheduling the activities with a higher priority as soon as possible without violating the resource and precedence constraints.

A lot of heuristic methods can be found in the literature and the following ones have been considered in this thesis to create a benchmark:

- **Shortest Imminent Operation (SIO)** It schedules the activities based on their duration, namely that the shorter the duration of an activity, the higher its priority to be scheduled.
- **Greatest Total Resource Demand (GTRD)** It schedules the activities on the basis of the total resource demand, namely that the higher the resource demand of an activity, the higher its priority to be scheduled. The idea behind this policy lies in scheduling potential bottleneck activities as soon as it is possible.
- **Greatest Resource Utilization (GRU)** It schedules the activities at time t to maximize the resource utilization within a predefined time horizon.

Those methods were first introduced for the RCPSP already by Patterson (1973) but then further used in many other later works such as, for instance, Adamu and Aromolaran (2018). Further information on heuristics and priority rules can be found in Kolisch and Hartmann (1999), Kolisch and Hartmann (2006) and Vanhoucke (2012).

One may notice that most of the literature on heuristic methods for the deterministic RCPSP is quite old. The reason is why the focus has moved either from

heuristics to meta-heuristics or from the standard deterministic RCPSP to other versions of the problem (multi-mode, multi-skill and so on) in the recent years.

Meta-heuristics

Meta-heuristics are general algorithmic frameworks, often nature inspired, designed to solve complex optimization problems (Bianchi, Dorigo, Gambardella, and Gutjahr (2009)). Applied to solve the RCPSP, they enhance and try to improve heuristic methods by effectively extending and exploiting the problem's search space (Abdolshah (2014)). In contrast to heuristic approaches, which generate RCPSP solutions with well-defined priority rules, meta-heuristic methods generate multiple solutions within the problem space while hoping to find a near-optimum solution (Agarwal, Colak, and Erenguc (2015)). Therefore, their underlying principle is to continuously create new solutions which become increasingly better as long as the search process continues. The search process finally terminates as soon as a certain stopping criteria is met. The most commonly used stopping criteria for the RCPSP in the literature is the maximum number of generated schedules. This criteria has the advantage that it makes it easier to compare different algorithms running on different hardware architectures and implemented with different programming languages. The drawback is that it does not consider the efficiency an algorithm generates a schedule with.

Some of the most noteworthy meta-heuristic algorithms applied to the RCPSP are listed below (Agarwal, Colak, and Erenguc (2015)):

- **Genetic Algorithms (GA).** They are by far the most popular meta-heuristic approach for optimization problems in general and the RCPSP in particular. As previously stated, GAs are inspired by the phenomenon of evolution of species observed in nature. In the evolution process, successive generations of populations of a species attempt to improve upon their previous generations through certain genetic and survival-of-the-fittest processes. For this reason, GAs are also called population-based meta-heuristics. When applied to an optimization problem, GAs employ similar processes to produce improved sets of solutions (generations of population) as the search progresses from one generation to the next. When applying GAs to the RCPSP, a set of activity lists acts as the population. From a given population of activity lists, a new population is produced through the reproduction process, involving an appropriate crossover mechanism.

The survival-of-the-fittest process is employed by being selective about the choice of parent activity lists used for producing a new offspring for the next generation of population. Valls, Ballestin, and Quintanilla (2008) are some of the most recent and relevant contributions of GA. Among all the meta-heuristics, the GAs have produced some of the best results in the literature for the RCPSP so far.

- **Simulated Annealing (SA).** The Simulated Annealing metaheuristics have been inspired by the process of annealing used in metallurgy for hardening metals. In the SA algorithm, a current solution is maintained at all times. Neighborhood solutions from the current solution are evaluated iteratively and, if a better solution is found, it becomes the new current solution. Occasionally, with a certain acceptance probability, a worse solution replaces the current solution as a mechanism to escape from a local neighborhood. The SA metaheuristics are stronger for local search than for a global search as shown in many works on this class of algorithms (Bouleimen and Lecocq (2003), Bouffard and Ferland (2007)).
- **Tabu search (TS).** Tabu Search employs intelligent use of memory to help exploit useful past experience in search. Memory is essentially a list of previously visited solutions. Several types of lists are maintained, each for a different purpose. A short-term tabu list includes recently visited solutions. Its purpose is to help avoid cycling within the same neighborhood. If a new solution is in this short-term list, it implies that the current neighborhood should not be explored further and the search should diversify to another neighborhood by using a different starting point. A list of poor solutions is also maintained so that if the search leads to a neighborhood of poor solutions, its presence can be detected and the search directed away from the current neighborhood. Tabu search based meta-heuristics for the RCPSP has been proposed by Thomas and Salhi (1998) and Nonobe and Ibaraki (2002).
- **Ant-Colony Optimization (ACO).** Ant colony optimization metaheuristics are inspired by the observed foraging behavior of ant colonies in which ants discharge a chemical substance called pheromone along the path between its colony and the food source. The smell of pheromone signals to the other ants in the colony the existence of previously followed paths. A stronger smell signals that a larger number of ants have been on that path more recently suggesting that food might be found there. After

some elapsed time, all ants in a colony figure out the shortest path towards food and they travel on the shortest possible path, thus optimizing their collective efforts. When applied to the RCPSP, the ant's paths are substituted by an activity lists and the pheromone signals are indicators of the goodness of placing an activity at a particular position in the activity list. This algorithm class was tested on the RCPSP, for example, by Merkle, Middendorf, and Schmeck (2002) with good results.

Although the best results in the literature for the RCPSP have been achieved with meta-heuristic methods, they are generally associated with a much higher computing time (2 or 3 orders of magnitude higher) than traditional heuristic methods. Moreover, in contrast to many heuristic methods they do not schedule the activities one after each other according to assigned priority values but they try to optimize the entire schedule at once. That makes them able to achieve better solutions in a purely deterministic scenario but makes the schedule prone to disruptions if even small deviations in the activity durations occur like it may happen in a real case. Moreover, other meta-heuristic methods relax this assumption by considering stochastic durations with known distributions (Ballestín (2007), Li and Womer (2015), Fang, Kolisch, L. Wang, and Mu (2015)). Again, in a real case this distribution may be unknown and their results are as good as the assumptions they are based on.

The assumption for this thesis, is that, even though the problem is modeled with deterministic times, a certain level of uncertainty is expected for the activity durations.

As a result, only algorithms with a reactive scheduling policy (see Section 2.5.4) will be considered for a comparison with the proposed machine-learning approach and their robustness will be assessed with a sensitivity analysis with different levels of uncertainty.

Machine learning

Machine learning approaches have almost never been used in the literature to solve the RCPSP. Only one contribution proposing machine learning approaches for the project scheduling problem could be found. In Adamu and Aromolaran (2018) machine learning was used to dynamically choose the best reactive scheduling policy among a set of predefined simple ones at every decision point. However, only very small projects with 11 activities and 13 different priority

rules were considered and there was no comparison with other algorithms. Nevertheless, Mao, Alizadeh, Menache, and Kandula (2016) has already tested a methodology based on machine learning in a similar problem with promising results and, as a consequence, this class of methods seems promising for further research contributions in the context of the RCPSP as well, like the one presented in this thesis.

2.6.3 Sensitivity analysis

Although the majority of the studies in the field of scheduling algorithms rely on deterministic approaches, projects face various sources of uncertainty: some activities may take more time than expected, some resources may become unavailable due to machine breakdowns and so on. The RCPSP is even characterized by the so-called Graham anomalies, namely that the total duration can increase if the duration of one activity decreases (Radermacher (1985)). In this context, the sensitivity analysis plays an important role.

However, as stated by Habibi, Barzinpour, and Sadjadi (2018), the majority of available studies in context of sensitivity analysis of scheduling problem have been conducted on machinery and workshop scheduling. Therefore, raising and answering similar questions for the project scheduling can be an interesting research field for future studies. One of the few contributions on this topic is Herroelen and Leus (2005) where the focus is on determining the permitted changing intervals of parameters, in a way that the optimality of full rescheduling is guaranteed by using simple modification techniques, such as right shift method.

In this thesis a different aspect of the sensitivity analysis will be analyzed, namely how the goodness of different reactive scheduling policies changes if the real activity durations deviates from the assumed ones.

2.7 Reactive scheduling with uncertain durations

When uncertainties about the activity durations come into play, it is still possible to compute the hypothetical starting time of each activity assuming deterministic durations. For the activities starting at $t = 0$, the starting time remains unchanged. However, since the deviations accumulate over time, the further in the future the activities have been scheduled, the more likely is that the starting

times and/or the scheduling order of the activities are no more feasible. In this case, a so-called schedule disruption occurs.

As a result, instead of an algorithm that computes a good à priori schedule, a reactive scheduling policy that decides at each decision point which activities should be started, may be preferable (Rostami, Creemers, and Leus (2018)). Decision points are typically the beginning of the project and whenever one or more activities have been just completed and, consequently, some resources become available again to start new ones. At each decision point t, a reactive scheduling policy can only use information that has become available up to t, together with à priori knowledge of the probability distributions of the project's parameters if known (Stork (2001)).

The main goal of this thesis is the design of reactive scheduling policies for the RCPSP problem where the activity durations incorporate a certain level of uncertainty and the author aims to do it using neuronal networks as scheduling tool.

2.8 Project instances

When it comes to measuring the performance of a project scheduling policy, a large number of activity sequences is required to consider a wide spectrum of decision situations and, as a result, to obtain a representative performance indicator. In the literature, a number of project libraries can be found and they normally differ in three main characteristics, namely the number of activities per project, the project generation tool used to create them and the parameters used for the generation. In most of the contributions in the field of the RCPSP, a preexisting project library is chosen and the new proposed scheduling policy is tested on its projects. The advantage is that other state-of-the-art algorithms have been likely tested on the same set of projects which makes the comparison very straightforward.

The performance of a generic scheduling policy on a project library is summarized by a single performance indicator PI which is normally the average gap between the total project duration using the considered scheduling policy and the one using the so-called lower bound critical path method (LB-CPM). The latter refers to a scheduling policy which generates a fictitious (unfeasible) schedule without considering the resource constraints which is equivalent

to assume an infinite number of available resources for each resource type. Although the LB-CPM refers to an unreachable lower bound, it provides a fast and easy way to compute a lower bound for the relative comparison among the state-of-the-art algorithms.

Considering a project library with P projects, equation 2.5 is generally used to compute the performance of a generic scheduling policy denoted as Π.

$$PI_{\Pi} = \frac{\sum_{p \in P} MS_{p,\Pi} - MS_{p,LB-CPM}}{P} \tag{2.5}$$

For what concerns the project libraries for the RCPSP, the first ones were proposed in Sprecher and Kolisch (1996) and their family is called Project Scheduling Problem Library (PSPLIB). The PSPLIB contains three separated project libraries denoted as J30, J60 and J120. The number after the letter J denotes the number of activities per project, while the number of resource types K is always equal to four. The projects within the same library are organized in groups of ten and for each group a different parameter combination has been chosen for the generation. The parameters which influence the project generation are the network complexity coefficient (CNC), the resource factor (RF) and an indicator on the so-called resource strength (RS). The CNC indicates the average number non-redundant edges per activity, the RF the average amount of resource types per activity and the RS provides information about the activity resource consumption. The formulas to calculate the three factors can be found in Kolisch (1996). For example, for the project sets J30 and J60 the following values has been considered: CNC = 1.5, 1.8, 2.1, RF = 0.25, 0.5, 0.75, 1 and RS = 0.2, 0.5, 0.7, 1. All possible combinations have been considered, i.e. 48 combinations, which results in 480 different projects. In the project set J120, 60 parameter combinations have been considered instead, resulting in 600 project instances.

The projects of the PSPLIB have been generated with a project generator called ProGen which takes the three mentioned parameters as an input and returns any number of project instances which comply with those parameters. Although the project libraries generated with the ProGen remain the most used ones in the literature, further project generators have been proposed in the following years, among which the RanGen (Demeulemeester, Vanhoucke, and Herroelen (2003)) and its extension RanGen2 (Vanhoucke, Coelho, Debels, Maenhout, and Tavares (2008)) are the most popular ones. The latter allows to produce

project instances by setting six different parameters and it has been used to create a project library with 1800 instances each with 30 activities known as RG30, which is often used in the literature (Artigues, Leus, and Nobibon (2013), Vanhoucke, Coelho, Debels, Maenhout, and Tavares (2008)).

2.9 Literature analysis and research gaps

The amount of literature contributions on the RCPSP is huge. Considering the basic RCPSP alone, i.e. the (single-mode) resource-constrained project scheduling problem with deterministic activity durations, renewable resources with known total availability, without preemption and with the total project duration minimization as goal, it is possible to find dozens of contributions. Nevertheless, some research gaps could be found. The following sections present a brief summary of the literature on the RCPSP and the research gaps that this thesis aims to fill.

2.9.1 Literature summary

The RCPSP problem has been presented in the 70s and the exact and heuristic methods were the first ones to be tested on it. While the exact methods provide optimal solutions only for small and easy projects since the computing time rapidly increases with the project complexity (activity number, resource type number and topology), the heuristic ones have been proven to achieve good results in a reasonable amount of time (Abdolshah (2014)).

Starting from approximately the beginning of 21^{st} century, meta-heuristic algorithms have been applied more and more on this problem with outstanding results. Figure 2.6 represents the performances in terms of computing time to schedule the activities and the gap from the lower-bound critical path method (LB-CPM) in terms of time units for different state-of-the-art algorithms on the so-called J120 project set (see Section 2.8), which is composed by 600 randomly generated projects with exponentially distributed activity durations. The real lower bound cannot be computed and, as a result, the lower bound critical path method (LB-CPM), which is the exactly computed unreachable lower bound that can be achieved if the resource constraints would be neglected, must be used for benchmark purposes.

In Figure 2.6 the following algorithms have been tested:

- **Random.** The priority values are assigned randomly.
- **Shortest Imminent Operation (Heuristics).**
- **Greatest Total Resource Demand (Heuristics).**
- **Shortest Imminent Operation (Heuristics).**
- **Greatest resource utilization 1 (Heuristics).** The combination of starting activities are chosen in a way that the resource utilization in a time horizon of 1 time unit is maximized.
- **Greatest resource utilization 5 (Heuristics).** The combination of starting activities are chosen in a way that the resource utilization in a time horizon of 5 time unit is maximized.
- **Greatest resource utilization 10 (Heuristics).** The combination of starting activities are chosen in a way that the resource utilization in a time horizon of 10 time unit is maximized.
- **Ballestin 2007 (Meta-heuristics).** Both with 5.000 and 25.000 generated schedules (Ballestín (2007)).
- **Ballestin and Leus 2009 (Meta-heuristics).**Both with 5.000 and 25.000 generated schedules (Ballestín and Leus (2009)).
- **Ashtiani 2011 (Meta-heuristics).** Both with 5.000 and 25.000 generated schedules (Ashtiani, Leus, and Aryanezhad (2011)).
- **Fang 2015 (Meta-heuristics).** Both with 5.000 and 25.000 generated schedules (Fang, Kolisch, L. Wang, and Mu (2015)).
- **Li 2015 (Meta-heuristics).** (Li and Womer (2015))
- **Rostami 2018 (Meta-heuristics).** Both with 5.000 and 25.000 generated schedules (Rostami, Creemers, and Leus (2018)).

The heuristic methods were presented in Section 2.6.2 and their performances, along with the ones for the random method, have been computed in this thesis, as no reference on this project dataset could be found in the literature.

Taking a look at figure 2.6, it is possible to make two considerations:

- There is a general trend, namely the higher the computing time, the smaller the makespan and, consequently, the gap from the lower bound LB-CPM.

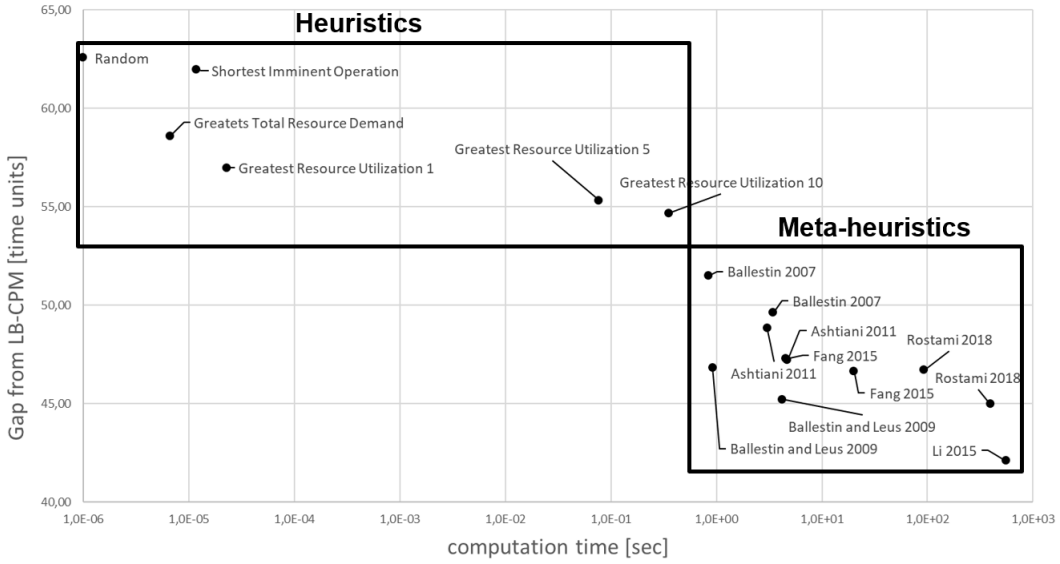

Figure 2.6: Performance and computing time of different scheduling algorithms.

- Although the meta-heuristic methods outperform the heuristic ones, they are always associated with a much higher computing time (approximately between a half and 8 orders of magnitude higher) than traditional heuristic methods.

2.9.2 Research gaps

As mentioned in Section 2.6.2, machine learning approaches have almost never been used in the literature to solve the RCPSP. This class of algorithm offers the possibility to design decision tools (e.g. a deep neural network model) which can learn to schedule the activities. If properly trained, they can learn to imitate the scheduling rules of the best state-of-the-art algorithms. Although they normally involve very high training times, they are able to process the input information and take decisions very fast (Walsh, O' Mahony, Campbell, Carvalho, Krpalkova, Velasco-Hernandez, Harapanahalli, and Riordan (2019)). As a result, they can potentially create scheduling tools which have better performances than the heuristic methods and much smaller decision times than the meta-heuristic ones.

The second research gap is related to the sensitivity analysis. Although the majority of the studies in the field of project scheduling rely on deterministic ap-

proaches, in real-world applications it may involve various sources of uncertainty such as deviations from the expected activity duration, resource consumption or resource total availability. As a result, it is important to investigate the robustness of the computed schedule or of the designed scheduling tool if the real unknown problem parameters deviate from the assumed ones. The majority of available studies on sensitivity analysis for scheduling problems have been conducted on machinery and workshop scheduling, while very few contributions on the RCPSP could be found.

2.10Projects in real-world applications

In the literature there is no systematic classification of real-world applications of what could be modeled as a Resource-Constrained Project Scheduling Problem. In fact the problem is most of the times studied as a pure optimization problem on randomly generated projects without providing a real use case. However, one could say that most of the real-world applications with the following characteristics are very suitable for such a modeling:

- A set of tasks must be performed.
- Some start-to-finish precedence constraints must be taken into account, namely that some tasks cannot start before others have been completed.
- The tasks use renewable resources which are available in a limited quantity and shared.
- The goal is to complete all tasks in the fastest way possible.

In real-world applications, the renewable resources can be many different things. Again there is no systematic classification of what can be modeled as a renewable resource but it is basically everything that can be used to perform a task with and it becomes again available once the task is finished. The following list provides some reasonable examples according to the author:

- **Manpower, workers, operators.** People are for sure one of the most commonly used resources to perform tasks that are manual or require manual intervention or supervision.
- **Tools.** Tool is a generic term to indicate an object used to extend the ability of an individual or a machine to modify features of the surrounding environment. More concrete examples could be tools used by operators in

a workshop (screwdrivers, welding machines and so on) but also specific machine tooling for milling, boring or turning.

- **Transporters.** In a production environment, the material flows from one working station to the other. Transporters like Automated Guide Vehicles (AGV), forklifts or cranes can be, for instance, modeled as resources (Pagani, Fischer, Farquhar, Skilton, and Mittwollen (2019)).
- **Locations and paths.** Transporters navigate in the production system and bring the material where it is required. As their number increases, it is possible that some traffic situations occur. To model this phenomenon, it is possible to consider different locations and paths as resources. By doing that, every time a task requires a transportation from one location to another, a route is chosen and the end location along with the required intermediate paths are reserved.
- **Computer resources.** In the era of digitization computers have assumed a crucial role in our world and they are more and more often involved in performing some tasks. As a result, computational resources such as CPU cores and GPU and memory resources such as RAM and data storage devices can be modeled as resources as well Mao, Alizadeh, Menache, and Kandula (2016).

Considering the two previous lists, it is now possible to better identify possible real use cases for the RCPSP. Some of them are taken from the literature, while others are just possible examples suggested by the author of this thesis:

- **Construction project management.** Bruni, Beraldi, and Guerriero (2015) is one of the few contributions that apply the RCPSP for the management of a real project. In this case, a construction project for university apartments was chosen and a schedule was suggested.
- **Product development management.** Product development requires a product to undergo different phases, namely the requirements specification, the product planning, development, process planning and production (Eigner and Stelzer (2009)). These phases can be divided in many subtasks which may run in parallel and may concur for the same resources.
- **Production and assembly scheduling.** Production systems are quite complex environments. Considering, for instance, matrix production, which is a concept of a flexible, flow-oriented, independent production system based on production cells (Hofmann, Brakemeier, Krahe, Stricker,

and Lanza (2018)), there are different products being produced in parallel and visiting the different working stations in different orders. The manufacturing operations could be modeled as project activities and the required resources could be the stations and the workers.

- **Management of maintenance activities.** Maintenance sequences for large assets normally require a large number of activities to be performed in a predefined sequence and with a limited number of resources. For example, Pagani, Fischer, Farquhar, Skilton, and Mittwollen (2019) presented an example of maintenance processes modeled with a framework which is an extended version of the RCPSP.
- **Job allocation for computer clusters.** Computer clusters are powerful computing devices which receive tasks to run from different users. Those tasks may have precedence relations among them and among their sub-tasks. In this case, the constraining resources are the computational resources, e.g. the CPU cores, and the storage memory (Mao, Alizadeh, Menache, and Kandula (2016)).

3 Deep neural networks

The only limit to AI is human imagination.
-C. Duffey

This chapter is dedicated to basic machine learning fundamentals that are necessary to comprehend the artificial neural network-based decision tools designed in this thesis and applied to the RCPSP. First of all, the idea behind machine learning is introduced along with the concept of deep neural networks. In the second part of the chapter, it is explained how their training and hyperparameter tuning work. Finally, some practical applications in decision-making processes from the literature are presented.

3.1 Introduction to machine learning

Machine learning as a domain of artificial intelligence denotes the ability to acquire knowledge by extracting patterns from raw data (Goodfellow, Bengio, and Courville (2016)). Therefore, machine learning involves the creation of learning algorithms that are able to learn from data while no explicit rules or logic are defined by a human (Khan, Rahmani, Shah, and Bennamoun (2018)). The dataset that is employed to make the algorithm learn is referred to as training set. According to Mitchell (1997), learning in the field of machine learning can generally be defined as follows: a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. In the following paragraphs, more information on task T, performance P and experience E is provided.

3.1.1 Definition of task T

Machine learning is mainly applied when a task is too difficult to be solved with a predefined algorithm created by humans (Goodfellow, Bengio, and Courville (2016)). The description of a machine learning task T is mostly guided by how the machine learning model is meant to process some input data which comprises a collection of features consisting in quantitative values processed by the machine learning model. Usually, the input data is represented as a vector $x \in \mathbb{R}^n$ where each element x_i of the vector represents a feature. Taking image data as an exemplary illustration, the values of the pixels represent its features. A collection of input data is called dataset and, in this case, it is represented by the set of available images.

One of the most common machine learning tasks which is at the same time relevant for this thesis is classification. In this type of task, the machine learning algorithm is configured to specify to which of k categories some input belongs to. This is employed by producing a function f:$\mathbb{R}^n \rightarrow 1,...,k$ (Goodfellow, Bengio, and Courville (2016)). In the special case of a binary classification task k=2 and the labels of the output y can for example be *yes* or *no*, while for a multi-class classification problem y entails k different labels (Khan, Rahmani, Shah, and Bennamoun (2018)).

3.1.2 Definition of performance measure P

In order to evaluate the capabilities of a machine learning algorithm, one has to determine one or several quantitative measures to assess the algorithm's performance. In most cases, the performance measure P is specific to the task T executed by the system. Regarding the performance of a machine learning algorithm, it is crucial to evaluate how the algorithm will perform on previously unseen data, since this determines how well it will perform when deployed in the real world on new situations. Unseen data is generated by creating a test set that is strictly separated from the training set and, thus, is never used to train the machine learning algorithm (Goodfellow, Bengio, and Courville (2016)). After the machine learning algorithm is trained on the training dataset, its performance is evaluated on the test dataset by computing performance measures P.

3.1.3 Definition of experience E

Machine learning algorithms can be categorized into supervised, unsupervised or reinforcement learning depending on what kind of experience they are provided with during the learning process (Goodfellow, Bengio, and Courville (2016)).

Supervised learning

Supervised learning algorithms use a training dataset that contains data points including a so-called label or target value. The training data have the form of a collection of pairs (data:x,label:y) where x represents the input of the machine learning model, i.e. the example or the vector of features, and y represents the target output in form of a label (Khan, Rahmani, Shah, and Bennamoun (2018)). The goal of supervised learning is then to create a trained model able to produce a prediction $\hat{y}$ for a previously unseen real world data point x. The parameters of the model allow to control the behavior of the learning algorithm and thus its prediction performance, which is generally measured by calculating the proportion of examples for which it produced the expected output. Providing training data to a machine learning algorithm so that it learns the mapping from input data to corresponding output labels is known as supervised training of a machine learning model (Bishop (2006)).

Unsupervised learning

Unsupervised learning algorithms use a training dataset containing many data points and learn useful properties from the structure of this dataset. In contrast to supervised learning, the outputs have no label or target values and, therefore, there is also no output prediction (Khan, Rahmani, Shah, and Bennamoun (2018)). The objective is usually to learn the structure of the dataset by learning the entire probability distribution that underlies the dataset. Typical unsupervised learning algorithms are used to perform tasks like clustering, where the goal is to separate the dataset into clusters of similar examples (Goodfellow, Bengio, and Courville (2016)).

Reinforcement learning

Apart from supervised and unsupervised learning there is a third category of machine learning which is reinforcement learning. In contrast to supervised and unsupervised learning algorithms, reinforcement learning algorithms do not experience a fixed training dataset but rather interact with an environment so that there is a feedback loop between the learning system and its experiences (Goodfellow, Bengio, and Courville (2016)). As a consequence, the system learns what to do by mapping situations to actions so that a numerical reward signal is maximized (Sutton and Barto (2018)).

In recent years, different classes of machine learning algorithms have been applied to scheduling problems similar to the RCPSP (e.g. Mirzaei and Akbarzadeh (2012) and Mao, Alizadeh, Menache, and Kandula (2016)). In this thesis, a supervised learning approach have been chosen.

3.2 Model training

This section focuses on presenting how to train a machine learning model so that the training process of the deep learning model applied in this thesis can be understood. Hence, in the following subsections, the important concepts of generalization, under- and overfitting, model capacity and hyperparameter tuning are outlined.

3.2.1 Generalization

As described in Section 3.1, the central objective of a machine learning model is to perform well with new and unseen inputs. This is known as generalization. How to estimate the generalization skills of a machine learning model is described below.

The training process consists in learning from training data while the error is monitored. One could think that the final goal is to achieve a model that has a low training error on training data so that it obtains good a performance. However, having a model trained sufficiently on the training set so that its training error is low does not guarantee that the model will provide good performance on previously unseen data, since the model has so far only experienced the

training data examples (Aggarwal (2018). Due to the fact that its true capabilities are determined on how well it performs on unseen data, an error measured on previously unseen data points is needed. This error is called test error or generalization error (Goodfellow, Bengio, and Courville (2016).

The expected test error is always greater than or equal to the expected training error value. How well a machine learning model is able to generalize and hence perform on unseen data is therefore determined by two abilities:

- making the training error small
- minimizing the gap between the training and test error

These two factors are related to two main challenges in machine learning which are underfitting and overfitting (Goodfellow, Bengio, and Courville (2016)).

3.2.2 Underfitting and overfitting

Underfitting occurs when a machine learning model is not able to achieve a sufficiently small training error, while overfitting occurs when there is a too large gap between training and test error. Figure 3.1, taken from Ketkar and S. (2017), schematically compares the training and generalization errors of a model with the minimum achievable error on the present task for an underfitting, ideal and overfitting scenario. In an ideal situation, training and generalization error are equal to the minimum achievable error. In the underfitting scenario, training and generalization error are greater than the minimum error, whereas in the overfitting scenario the training error is small, however, the generalization error is big.

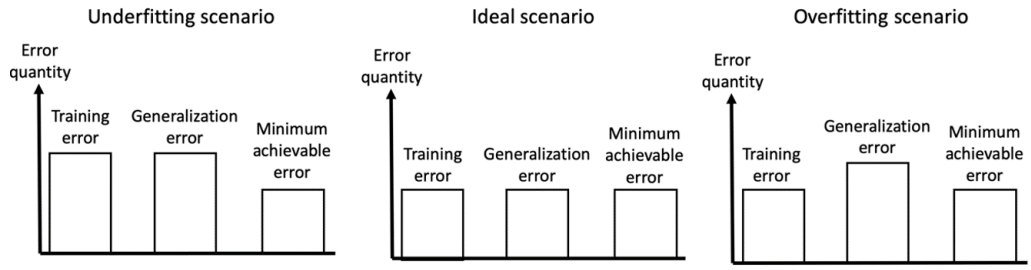

Figure 3.1: Comparison of training, generalization and minimum achievable error.

For what concerns the performance, a model that overfits obtains a good performance on the training data but fails to generalize well to unseen data (Khan,

Rahmani, Shah, and Bennamoun (2018)). On the other hand, an underfitted model achieves poor performance on both training and test data.

There are several reasons why a model can underfit or overfit and perform badly. Some of them are, for instance, too few training data, too few training epochs, low model capacity and so on. Additional information on the latter are provided in the following section.

3.2.3 Model capacity

Informally, the capacity of a model is determined by its ability to model complex relations between inputs and outputs. Models with a low capacity are not expressive enough to fit to the training set and thus underfit. On the other hand, a model with a too high capacity is more prone to overfit, which means poor performance on unseen data, although the performance on the training data are good. In order to avoid over- or underfitting, the capacity of a machine learning model needs to be adapted appropriately to the true complexity of the machine learning task and the amount of training data provided (Goodfellow, Bengio, and Courville (2016)). A model with an insufficient capacity is unable to solve complex tasks. On the contrary, a model with a high capacity can solve complex tasks but tends to overfit if the capacity is higher than needed to solve the present task. Since models with a higher capacity usually have more parameters, those models require also more training data to obtain a good generalization power on unseen data (Aggarwal (2018). The typical relation between capacity and error is that, with an increasing model capacity, the training error decreases until it reaches an asymptote corresponding to the minimal possible training error. The generalization error usually follows a U-shaped curve as a function of the underlying model capacity. This typical relation between capacity and error is illustrated in Figure 3.2 according to Goodfellow, Bengio, and Courville (2016):

At the left end of the graph where a model's capacity is very low, training and generalization error are both high. This corresponds to an underfitting scenario. With an increasing model capacity, the training error decreases but the gap between training and generalization error increases. In any case, the gap can be always reduced by using more training data for the training process (Goodfellow, Bengio, and Courville (2016)).

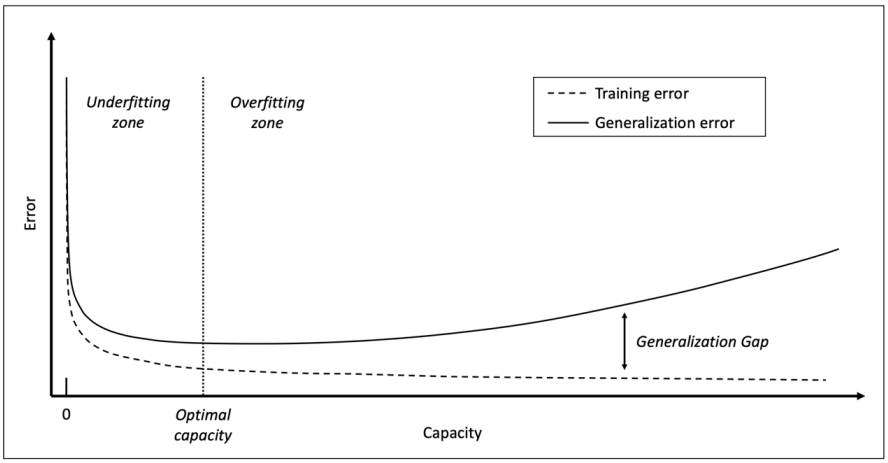

Figure 3.2: Relation between model capacity and error.

The next section covers details on how the so-called hyperparameters of a model, i.e. the model parameters that cannot be learned during the training, can be used to influence the capacity of the model and how they can be tuned to achieve a better model performance after the training.

3.2.4 Hyperparameters

Machine learning models are defined by a set of hyperparameters that control the algorithm's behavior (Claesen and De Moor (2015)). The hyperparameters have a direct influence on the capacity of the underlying machine learning model and, as a result, they are employed to adjust a model's capacity to the complexity of the present task. In contrast to the parameters that are considered in the mapping function of the machine learning algorithm, hyperparameters are not learned during the training process but must be set by the user before it (Goodfellow, Bengio, and Courville (2016)). If they are set properly, the risk of under- and overfitting of a model can be drastically reduced (Claesen and De Moor (2015)).

In order to visualize the effect of varying the capacity of a model by modifying its hyperparameters, Figure 3.3 shows three different polynomial regression machine learning models that were created by fitting different functions to an exemplary training dataset. The dataset exhibits a natural quadratic course in a two-dimensional space. A polynomial regression exhibits a single hyperpa-

rameter which is the degree of the polynomial, while the coefficients are learned during the training. The example in Figure 3.3 is taken from Goodfellow, Bengio, and Courville (2016).

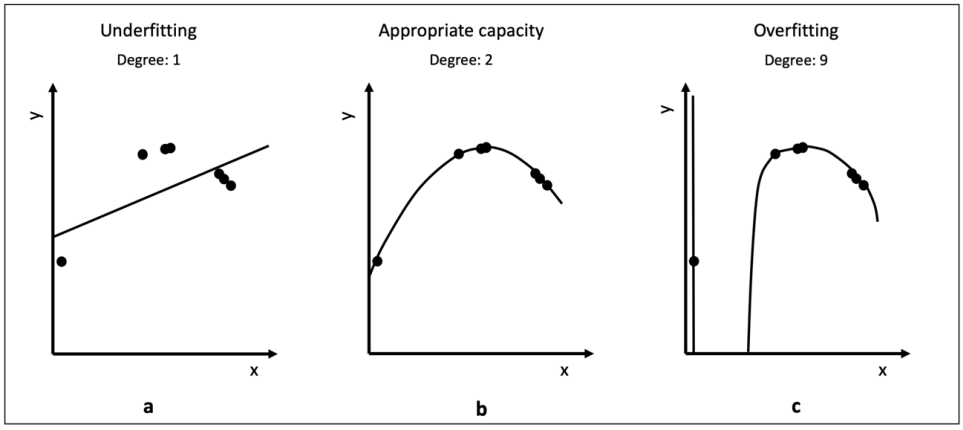

Figure 3.3: Polynomial regression with fitting machine learning models of different capacity.

In the first graph, the value of the polynomial degree is 1 and a linear function models the relation between the inputs x and the outputs y of the dataset. The linear function suffers from underfitting since it cannot capture the curvature that is present in the data. In the second graph, a second degree polynomial is used and it shows a quadratic function that fits well the data and is able to generalize appropriately. It does neither suffer from overfitting nor underfitting. In the last graph a polynomial of degree 9 is used to fit the data and thus overfitting occurs. Only when the capacity of a model is set appropriately, it is able to correctly generalize and thus provides a good performance. Consequently, in order to maximize the performance of a machine learning model, it is crucial to choose the right capacity and consider that hyperparameters have an effect on the model's performance on the present task.

The process of finding appropriate hyperparameters for a machine learning model in order to maximize its performance is often called hyperparameter tuning or hyperparameter selection and is explained in the next section.

3.2.5 Hyperparameter tuning

So far, the training process of a machine learning model has been characterized by training a machine learning algorithm on a training set, followed by the evaluation of its generalization power by calculating the model's generalization error as well as observing the performance on the unseen test dataset. Splitting a dataset into a training and test set ensures that the data objects of the test set are never used to make any choices about the model, since it would lead to overfitting and a poor performance on new unseen data (Aggarwal (2018)). Similarly, any decision making about the hyperparameters of the model must never be done based on the test set, otherwise the model becomes influenced by the knowledge from the test set (Goodfellow, Bengio, and Courville (2016)). Hence, additionally to the training and test dataset, a third dataset for making choices about the hyperparameters is necessary, which is called validation set.

The hyperparameter tuning is performed in such a way that the same machine learning model is trained using the same training data and, afterwards, tested on the validation set several times with different hyperparameter configurations. The same training dataset can be used multiple times for different hyperparameter configurations or even completely different machine learning algorithms. This allows to compare the relative performance of different model configurations or algorithms. The objective of this process is to find the best performing model configuration, which is why this is also referred to as model selection. However, it is crucial, that the actual performance evaluation of the different model configurations is never conducted on the test set but only on the separate validation set. Otherwise, the hyperparameters and the model choice could lead to overfitting. The test set is only used once at the end of the tuning process to test the performance of the model with the chosen hyperparameter configurations to decide whether the model is good enough for a real world deployment. The insights from the results of this final evaluation on the test data must not be used for further adjustment of the model, otherwise the further results would be contaminated with the knowledge from the test set.

Different ratios for the division of the available training data in the three sets can be found in literature. As shown in Figure 3.4, Aggarwal (2018) propose a 50:25:25 split as a conventional rule, however, this should not be viewed as a strict rule and also other proportions exist in literature. Goodfellow, Bengio, and Courville (2016) assign 80 percent of the data to the training set and 20

percent to the validation and test set together. Ketkar and S. (2017) suggests devoting 60 percent of the data examples for the training set and keeping 40 percent for both the validation and test set equally split. Eventually, the split ratio can also be influenced by the size of the available dataset. Usually, the greater the size of the dataset, the higher the percentage of the data examples that can be assigned to the training set (Aggarwal (2018)). As a result, since a data generator tool is used in this thesis and any number of training data can be generated, a 80:10:10 split was used in this thesis.

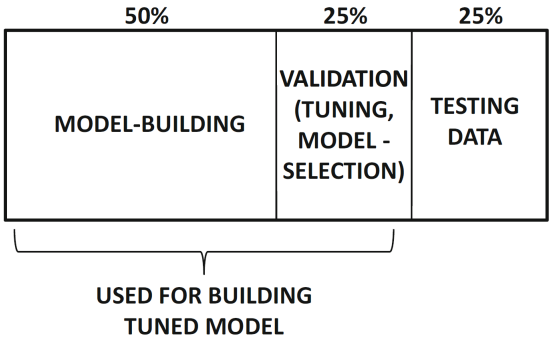

Figure 3.4: Training, validation and test data split.

3.2.6 Bayesian optimization to support the hyperparameter tuning

There are two general approaches to perform the hyperparameter tuning, namely the manual and automatic hyperparameter tuning. Usually, manual hyperparameter tuning requires a rough understanding of how the different hyperparameters of a model influence its capacity and thus its performance on the present task. In contrast, automatic hyperparameter tuning methods require less knowledge, since they automatically define which hyperparameter configuration should be tested in the next iteration (Goodfellow, Bengio, and Courville (2016)) which also increases the number of tested configurations in the same amount of time, since no manual intervention is required between the test of two consecutive configurations. Nevertheless, the user still has to define the boundaries of the search space.

Since no neural network applications on the RCPSP could be found in the literature, there are no existing guidelines on the hyperparameter choice. As a consequence, an automatic hyperparameter tuning has been implemented.

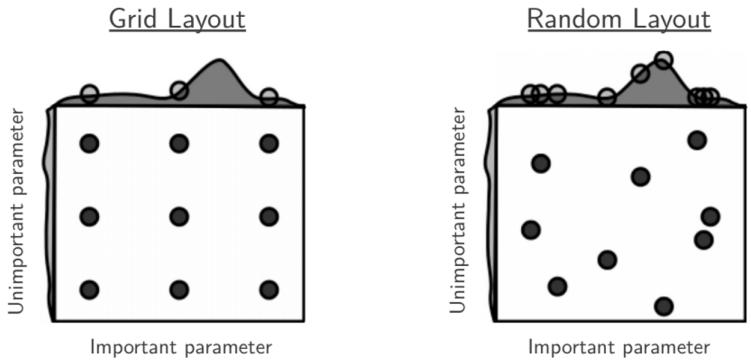

Figure 3.5: Grid and random search with nine trials.

As stated in Snoek, Larochelle, and Adams (2012), three possible optimization algorithms can be chosen to search for the best hyperparameter configuration, i.e. the one maximizing the model performance on the test set after the training:

- **Grid search.** This method divides the predefined interval of each parameter in equally spaced sub-intervals and only their middle points are considered as possible values in the evaluation. Considering the optimization of a problem with two parameters both with three sub-intervals, a total number of $3 \cdot 3 = 9$ parameter combinations are considered (see Figure 3.5 from Bergstra and Bengio (2012)). However, in a machine-learning model the number of hyperparameters and the number of considered values for each of them can be much bigger and, as a result, a lot of configurations must be tested. On the other hand, it can be easily parallelized.
- **Random search.** This method takes random values for the predefined interval of each parameter and tries out a bunch of randomly generated parameter configurations. As for the grid search, it can be easily parallelized and the maximum number of samples can be defined as a stop criterion. It generally gives a better performance than grid search (Snoek, Larochelle, and Adams (2012)) but there is the risk that some good regions of the search space are not considered.
- **Bayesian optimization.** In contrast to the two previous search algorithms, which use brute-force to test different parameter configurations, Bayesian optimization uses knowledge of previous iterations of the algorithm to select the parameters to test in the next iteration. Bayesian methods attempt to build a function that estimates how good the objective function might be for each set of parameters. By using this approx-

imate function, also called surrogate function, it is no more necessary to test many parameter configurations, since one can just optimize the hyperparameters with the surrogate function. The latter is estimated and updated at each iteration based on the results of all evaluations and, as a result, the more evaluations have been performed, the more accurate this function gets. Figure 3.6, taken from Brochu, Cora, and De Freitas (2010), represents an example of a Bayesian optimization with one parameter x. The dashed function is the unknown objective function. The goal of the bayesian optimization is to find the maximum value over x. The continuous black line represents the mean value of the surrogate function μ (estimation of the real objective function), while the dispersion region around it represents the uncertainty σ. The first picture represents the initial situation where two values for the parameter have been sampled. The next point to sample is chosen using the acquisition function which weights the expected advantage of sampling in an unexplored region and the one of sampling in a region where good results have been found in previous iterations.

The only drawback of this method is that it cannot be parallelized since the tested parameter configuration of one iteration depends on values of the objective function of the previous ones. However, many contributions in the literature agree that Bayesian optimization should be preferred to the other methods due to its searching efficiency (Snoek, Larochelle, and Adams (2012)). For what concerns machine learning algorithms, the generalization performance, i.e. the objective function to maximize, is modeled as a sample from a Gaussian process. The tractable posterior distribution induced by the Gaussian process leads to an efficient use of the information gathered by previous experiments, enabling optimal choices about what parameters to try in the next iteration (Pelikan, Goldberg, and Cantu-Paz (1999), Snoek, Larochelle, and Adams (2012)).

3.3 Introduction to deep neural networks

In this section, the fundamentals of deep neural networks and deep learning are covered. First of all, the concept of deep learning along with deep neural networks (DNN) are introduced. Afterwards, their learning mechanism is ex-

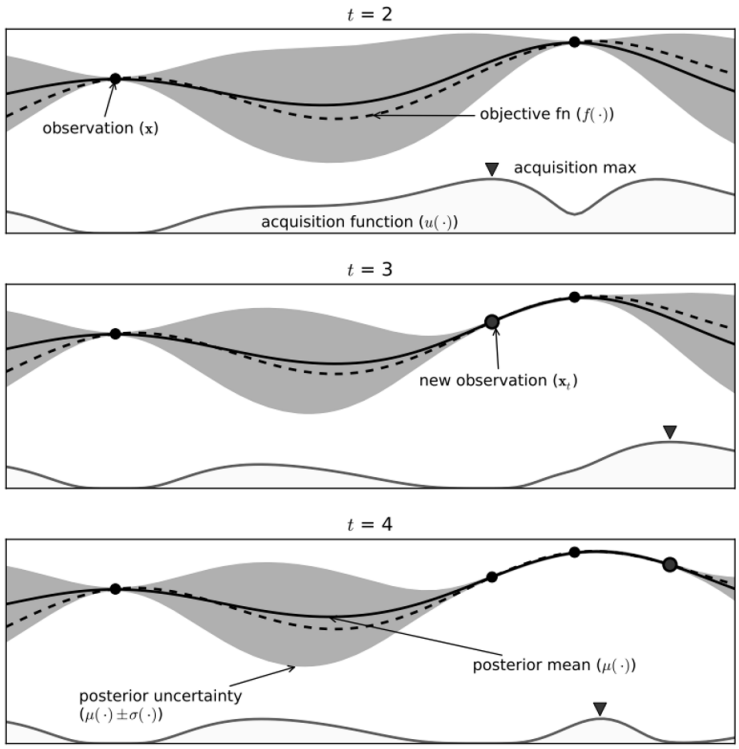

Figure 3.6: Example of Bayesian optimization with one parameter.

plained. Finally, the three different classes of neural networks that are used in this thesis are introduced, namely fully connected, 1-dimensional convolutional and 2-dimensional convolutional neural networks.

3.3.1 From machine learning to deep learning

Since traditional machine learning algorithms fail to generalize well on important and challenging artificial intelligence tasks like object recognition or speech recognition, the development of new concepts that are able to cope with these tasks has been driven forward in the past years. At the same time, the recent progress in developing more powerful computing devices and a greater availability of data has further accelerated the progress in the domain of artificial intelligence (Aggarwal (2018)). As a result, the development of deep learning concepts has been driven forward tremendously in the last few years. These deep learning concepts are artificial neural networks. The term *deep* comes from their depth through the stacking of computational layers (Goodfellow, Bengio, and Courville (2016)). The previously specified concepts of machine learning are

general concepts and thus also maintain their relevance for any kind of deep learning approach. Artificial neural networks as an essential concept of deep learning are outlined in the following section.

3.3.2 Artificial neural networks

Artificial neural networks originate from the work of McCulloch and Pitts (1943) who laid down the foundation for further research in the field of neural networks. Artificial neural networks mimic human biological neural networks by having many small interconnected computational units called neurons. In the following, the term neural network is always referring to an artificial neural network. There exist many different variations of neural networks that have their own peculiarities and domains of application (Aggarwal (2018)). However, to keep this thesis within its scope, only the fully connected and the convolutional network types are presented. Moreover, for both neural network types only feed-forward network configurations are used, namely neural networks with an acyclic structure and where the information moves in only one direction (Da Silva, Spatti, Flauzino, Liboni, and Reis Alves (2017)). Before outlining the fully connected neural networks in this section, the basic structure with a single neuron is described.

3.3.3 Perceptron

A single neuron has one output node and a single input layer and is also referred to as perceptron (Da Silva, Spatti, Flauzino, Liboni, and Reis Alves (2017)). An illustration of a neuron is depicted in Figure 3.7 which is taken from MacKay and Mac Kay (2003)).

A neuron has a number I of inputs x_i and one output y. Each input to the neuron is associated with a weight w_i with $i = 1, ..., I$. There is also an additional parameter w_0 called bias, which is associated with a certain input value Θ_P (MacKay and Mac Kay (2003)). The inputs are contained in input nodes of the input layer while no calculations are performed in this layer. The input values are processed in the neuron itself with a two-step procedure. Firstly, in response to the inputs x_i the activation value a of the neuron is calculated as in equation 3.1.

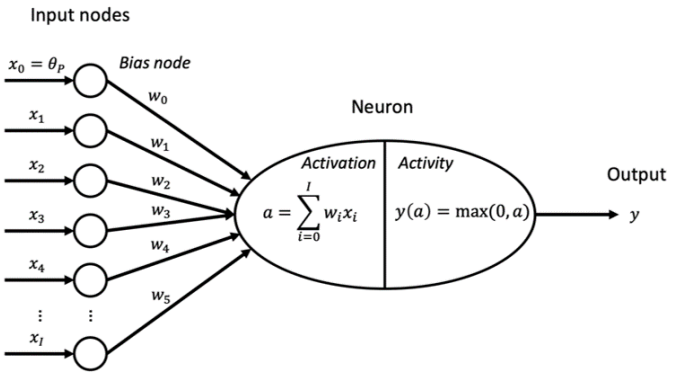

Figure 3.7: Representation of a single neuron.

$$a = \sum_{i=0,\ldots,I} w_i x_i \tag{3.1}$$

Secondly, the actual output y of the neuron is given by the function $f(a)$ of the activation value a. This function is referred to as activation function and there are several possible activation functions that can be deployed in a neuron. The most common activation functions are:

- The rectified linear unit function (ReLU) $y(a) = max(0, a)$ (**Nair2010**)
- The sigmoid curved logistic function $y(a) = \frac{1}{(1+e^{-a})}$
- The sigmoid curved tanh function $y(a) = tanh(a)$.

The three different activation functions are illustrated in Figure 3.8.

3.4 Fully connected neural network

A multilayer perceptron is a neural network that contains multiple computational layers between an input layer and an output layer. The intermediate computational layers contain neurons and are referred to as hidden layers, since the computations performed in these layers are generally not visible to the user (Da Silva, Spatti, Flauzino, Liboni, and Reis Alves (2017)). The default architecture of a multilayer perceptron assumes that all units of one layer are connected to each unit of the successive layer which is why those layers are also referred to as fully connected (Aggarwal (2018)). An exemplary illustration of a feed forward network is depicted in Figure 3.9 taken from MacKay and

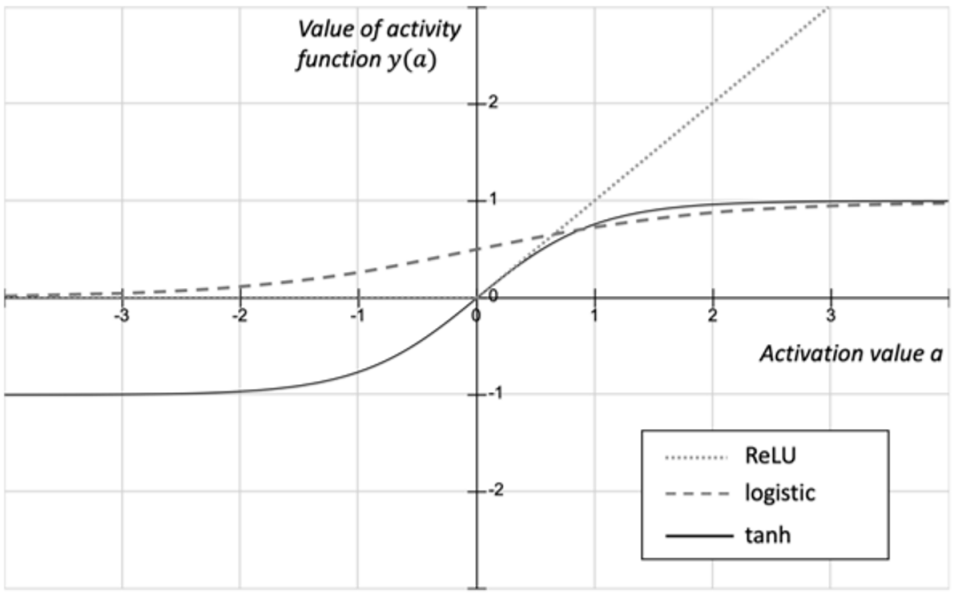

Figure 3.8: Different activation functions: ReLU, sigmoid logistic and tanh.

Mac Kay (2003). The depicted neural network consists of an input layer on the left and an output layer on the right. In between there are two hidden layers that consist of three neurons each. The neurons that contain the bias values for each layer are depicted separately.

3.4.1 Computational layers of fully connected neural networks

The main objective of an artificial neural network is to approximate a function f^* that maps all possible input values x to some expected output y (MacKay and Mac Kay (2003)), i.e. $y = f^*(x)$. A neural network defines the mapping $y = f(x, w)$ by learning the values of the weights w that result in the best function approximation (Goodfellow, Bengio, and Courville (2016)). The weights are also referred to as the parameters of the neural network.

In a feedforward configuration, the fully connected neural network is characterized by a chain structure where the neuron inputs of every layer are a function of the neuron outputs of the previous layer. The output of an intermediate layer l is given, for instance, by equation 3.2, while $h^{(l-1)}$ is replaced by x for the first layer.

$$h^{(l)} = y^{(l)}(W^{(l)T} \cdot h^{(l-1)} + b^{(1)}) \tag{3.2}$$

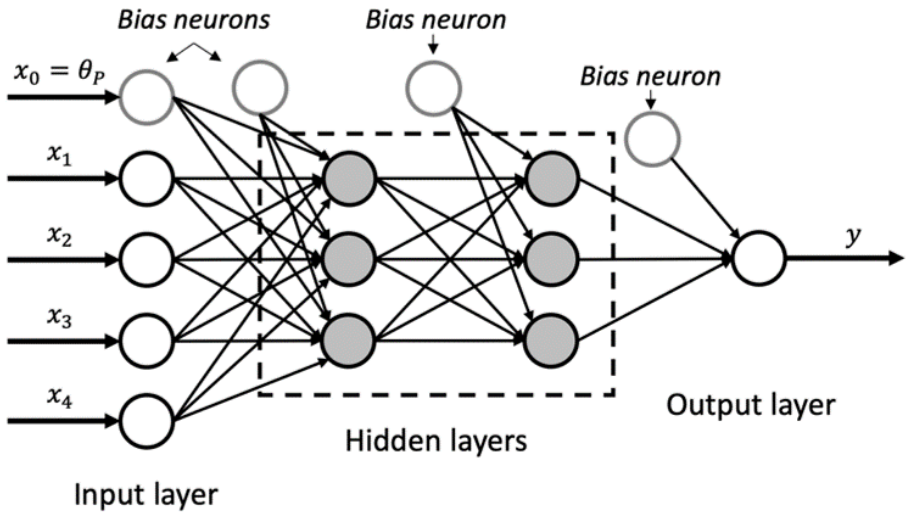

Figure 3.9: Multilayer perceptron.

In these formulas, y represents the activation function, b the bias values and W the weights of each layer. The h stands for the output that the layer produces. The number of layers gives the depth of the network while the number of units in a layer determines its width (Goodfellow, Bengio, and Courville (2016)).

3.4.2 Hyperparameters of fully connected neural networks

As mentioned in the previous paragraph, the hyperparameters that primarily control the capacity of a fully connected network are the number of hidden layers and the number of units or neurons per layer. A network with more hidden layers and more units per layer has a higher capacity. However, if the number of hidden layers or units is too large for the underlying machine learning task, the neural network model may overfit and thus perform poorly. Hence, the hyperparameters of the neural network need to be selected according to the underlying task so that it generalizes well and achieves a good performance, as it is also described in Section 3.2.4.

Neural networks usually perform well when their capacity is high while at the same time the tendency of overfitting is avoided by employing so-called regularization techniques (Goodfellow, Bengio, and Courville (2016)). Regularization can be regarded as any modification to the model or its training process with the objective of reducing the error on unseen data (Ketkar and S. (2017)). There

are plenty of regularization techniques available but only dropout and early stopping are introduced, since they are applied in the later part of the thesis. A compact overview over other regularization methods can be found in Khan, Rahmani, Shah, and Bennamoun (2018).

One of the most popular regularization strategies is the so-called "dropout" which has been introducted by Hinton, Srivastava, Krizhevsky, Sutskever, and Salakhutdinov (2012). During the training process of a neural network, each hidden unit is randomly omitted from the network with a certain probability, while the complete model is still used during the validation and testing phase. This has the effect of making the layer be treated like a layer with a different number of nodes and connectivity at different training steps. As a consequence, each update to a layer during training is performed with a different view of it. Dropout makes the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs. The dropout probability is usually set to 50% (Hinton, Srivastava, Krizhevsky, Sutskever, and Salakhutdinov (2012)).

Early stopping is another regularization method that is able to avoid overfitting. During the training phase of the neural network model, the model's performance on the validation set can be periodically monitored every fixed number of epochs. Once the performance of the model on the training set does not improve any more or even drops, the training process is stopped (Khan, Rahmani, Shah, and Bennamoun (2018)).

3.5 Convolutional neural networks

This section provides the fundamentals to understand how convolutional neural networks work in contrast to the fully connected ones. The biologically inspired convolutional neural network architecture was originally developed for computer vision and image classification tasks by Hubel and Wiesel (1959). Regarding these tasks, convolutional neural networks are the most successful neural networks and regularly prove their outstanding capabilities in various image recognition challenges like the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Krizhevsky, Sutskever, and Hinton (2012)). Even though convolutional neural networks are mainly applied to image data related tasks, they generally work quite well with any grid-structured input data, like sequen-

tial, temporal and spatial data (Aggarwal (2018)). The grid-structured inputs can be 1-dimensional (e.g. time-series) or 2-dimensional (e.g. images).

In contrast to the fully connected neural networks that exclusively consist of fully connected layers, convolutional neural networks have at least one layer that applies a mathematical operation called convolution. The convolution operation in convolutional layers is a dot-product operation between a grid-structured input x and a grid-structured set of weights w on different spatial positions of the input grid: $s = x \cdot w$ (Aggarwal (2018)). The weights w are referred to as filters or kernels and the output s of the convolution operation is called feature map. The weights w of the kernels are the learnable parameters in the convolutional layer. Within each convolutional layer, the processed grid structure is of a 3-dimensional nature since it has a height width and depth. It is important not to confuse the depth of the grid in a convolutional layer with the depth of the overall neural network. The depth of a convolutional layer represents the number of channels in the layer, such as, for example, the number of primary color channels (e.g., red, green, blue) of an input image or the number of feature maps that represent the output of the convolutional layer (Aggarwal (2018)). Based on the width and height dimensions of the input grid, it can be differentiated between 1-dimensional and 2-dimensional convolution operations. Figure 3.10 provides an example of how the convolution operation is performed with 1-dimensional and 2-dimensional filters and grids. The depth of both grids is 1. For the 1-dimensional convolution operation, it is explicitly illustrated how the convolutional kernel slides stepwise over the width of the input grid. For the 2-dimensional grid the filter slides both along the width and height of the grid. This figure has been taken from Ketkar and S. (2017)).

Similar to a fully connected layer, in a convolutional layer the output s is used by an activation function f. Thus, the overall output y of a convolutional layer can be expressed as $y = f(s(x \cdot w))$ with f representing the activation function. Equivalent to fully connected layers, the same variety of activation functions can be applied in convolutional layers (Khan, Rahmani, Shah, and Bennamoun (2018)). However, due to its computational effectiveness, the ReLU function is currently the most widely used activation function in convolutional layers (Krizhevsky, Sutskever, and Hinton (2012)). Just like for the feedforward neural network, the weights are initialized with values of a certain distribution prior to the training process and the backpropagation algorithm for the training is applied (Aggarwal (2018)).

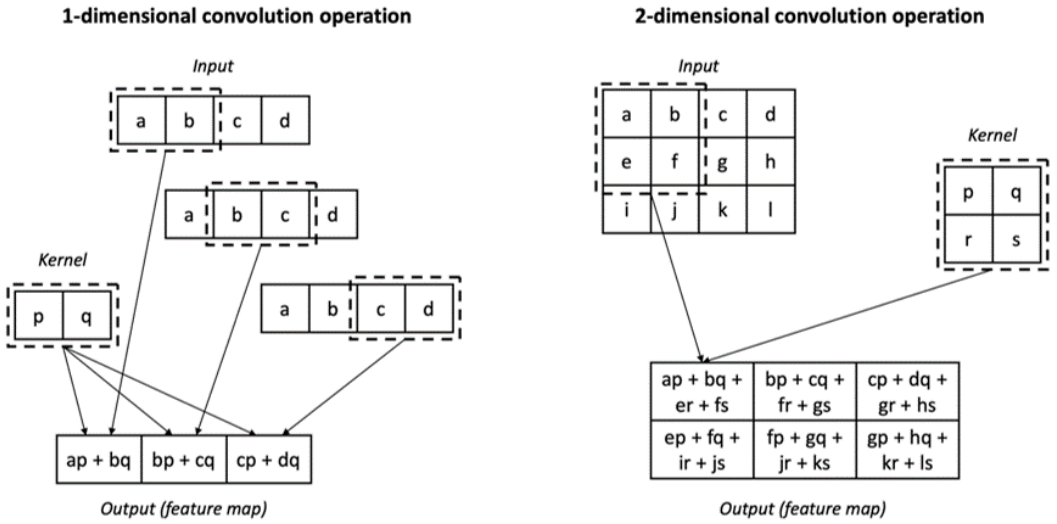

Figure 3.10: Convolution operation for 1-dimensional and 2-dimensional grids.

Convolutional layers have sparse interactions since the convolutional kernel is significantly smaller than the input information. This implies a reduction of the parameters to be stored and of the number of operations required to compute the output values. However, they are still capable of extracting useful features from their inputs. Since a convolutional neural network that has a comparable size to a fully connected network has fewer connections and parameters, it can be easier trained but the theoretically-best performance is usually only slightly worse (Krizhevsky, Sutskever, and Hinton (2012)).

3.5.1 Computational layers of convolutional neural networks

This section introduces neural networks with convolutional layers. A typical convolutional layer in a convolutional network consists of three stages: The first stage involves several parallel convolution operations performed on the input in order to produce a set of linear activation values. These convolution operations are usually performed with several different filters so that different features in the input grid can be detected (Aggarwal (2018)). The number of filters that are applied on the input grid defines the number of feature maps and hence, also the width of the layer. In the second stage, called detector stage, all generated activation values are run through the nonlinear activation function like, for example, the ReLU function (see Section 3.3.3). The final stage of a convolutional layer comprises a pooling function that further modifies the output

of the detector stage by grid-wise summarizing the output using some summary operation. In the context of convolutional layers, the most widely used max pooling operation returns the maximum output of a grid-shaped neighborhood. Similar to the kernels of the convolution operation, the rectangular grid of the pooling function is slidden across the grid of the output of the detector stage. In general, the pooling is employed to make the representation of the layer approximately invariant to small translations of the input. The different stages of a convolutional layer are displayed in Figure 3.11, which is taken from (Goodfellow, Bengio, and Courville (2016)).

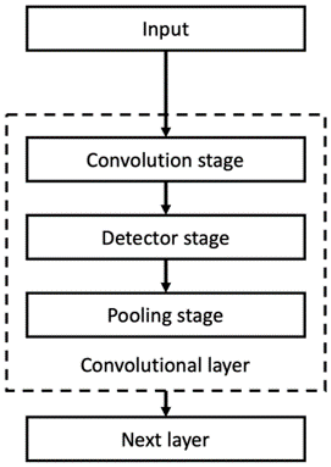

Figure 3.11: Stages of convolutional layer.

3.5.2 Hyperparameters of convolutional neural networks

Convolutional layers also have hyperparameters that influence the model's capacity and need to be set by the user prior to the training so that the model ideally fits the underlying task. In this section, the most relevant hyperparameters of convolutional neural networks are presented.

Convolutional filters

Some of the hyperparameters are related to the convolutional filters because their shape as well as their number can be varied. Increasing the width of the convolutional filters leads to an increase of the number of parameters and therefore also a gain in the model capacity (Goodfellow, Bengio, and

Courville (2016)). However, the spatial extents of the filters are usually oddly sized and in most cases do not exceed the dimensions of 3x3, 5x5 or 7x7-sized filters (Aghdam and Heravi (2017)). This is also demonstrated by the fact that the great majority of the state-of-the-art convolutional neural network architectures stick to these filter dimensions. There are two reasons for relatively small sized kernels. Small filters ensure that the number of learnable parameters remains manageable and facilitate the learning of distinctive patterns from many local regions in the input grid (Khan, Rahmani, Shah, and Bennamoun (2018)).

Increasing the number of filters in a convolutional network is another alternative for varying a convolutional neural network's capacity. The more filters are used, the more parameters are incorporated in that layer and thus the capacity is increased. Moreover, increasing the number of filters in one layer also increases the number of feature maps in the following layer as Figure 3.12 shows (Aggarwal (2018)). Using many different filters in a convolutional layer is motivated by the fact that each filter tries to identify particular types of spatial patterns in small rectangular regions of the input. Hence, the more the filters, the more different spatial patterns can be extracted from the input. In practice, later convolutional layers typically tend to have smaller kernel sizes, but greater depth in terms of feature maps. Additionally, the number of filters in each layer is mostly a power of 2, because this usually results in a more efficient processing (Aggarwal (2018)).

Strides

In Figure 3.10, the convolutional kernels were slidden horizontally and vertically by a step size of 1 at a time across the input grid. The step size, in this example equal to 1, is referred to as the stride of the kernel and can be set to any value. The higher the stride is set, the smaller the feature map of the output of the convolution operation gets. Hence, it reduces the dimensions which provides a moderate invariance to scale and pose of features in the input grid (Khan, Rahmani, Shah, and Bennamoun (2018)). For a given convolutional kernel with size $k \times k$, an input grid with size $h \times w$ and a stride length s, the dimensions of the output feature map is given by:

$$h' = [\frac{h - k + s}{s}] \tag{3.3}$$

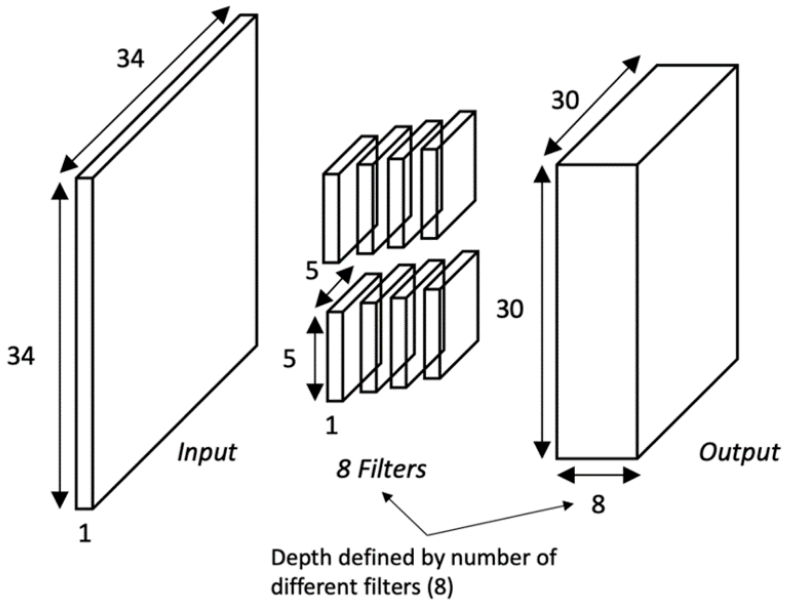

Figure 3.12: Effect of the number of convolutional filters on the layer's output.

$$w' = [\frac{w - k + s}{s}] \tag{3.4}$$

Padding

Since every convolution operation leads to a shrinkage of the output feature map, the number of sequentially arranged convolutional layers in a neural network architecture is limited to a point where the output map becomes too small (Goodfellow, Bengio, and Courville (2016)). The rate of shrinkage is especially dramatic when the kernel and stride sizes are large or the input grid is small. However, in order to be able to design deep neural networks with many convolutional layers, a quick collapse of the feature dimensions has to be avoided. A solution to this is the application of zero-padding around the input feature map, which essentially increases the horizontal and vertical input dimensions by adding a certain number of zeros to all edges of the input grid. If p denotes the number of zeros by which the input map is increased along each dimension, the dimensions of the output feature maps can be described as follows (Khan, Rahmani, Shah, and Bennamoun (2018)):

$$h' = [\frac{h - k + s + p}{s}] \tag{3.5}$$

$$w' = [\frac{w - k + s + p}{s}] \tag{3.6}$$

Max-Pooling

Lastly, the pooling stage of convolutional layers also has some hyperparameters that influence the model and need to be set by the user. The max-pooling operation is similar to the convolution one in having a rectangular grid sliding over its corresponding input grid. Therefore, the size of the pooling grid and the stride of the slides are hyperparameters that need to be specified by the user (Khan, Rahmani, Shah, and Bennamoun (2018)). Typically, the size of the region on which pooling is performed is 2×2 while the stride mostly has values of $s = 1$ or $s = 2$ (Aggarwal (2018)).

3.5.3 Typical design principles for convolutional neural networks

Due to the big number of hyperparameters, the design of a powerful neural network architecture that achieves a good performance on the underlying task can be challenging. However, ongoing research in the field of convolutional neural networks and the different successful convolutional neural networks applied in the ImageNet Challenge (Krizhevsky, Sutskever, and Hinton (2012)) have coined a few design principles for convolutional neural networks that could be extended to other applications. Therefore, these design principles have been also tested for the neural networks of the RCPSP solution approach proposed in this thesis.

First of all, a convolutional neural network typically consists of several convolutional layers followed by a few fully connected layers and an output layer (Aghdam and Heravi (2017)). Secondly, the number of convolutional filters usually increases with the depth of the convolutional neural network, hence, early convolutional layers have fewer kernels than later layers (Aghdam and Heravi (2017)). As observed by Goodfellow, Bulatov, Ibarz, Arnoud, and Shet (2013), increasing the number of parameters in the convolutional layers without increasing the depth of the network is not as effective as having many parameters in the layers of a deep network. Thus, convolutional layers with many parameters should always be embedded in a deep network to exploit their potential. This has also been confirmed by the recent winning convolutional neural networks of the Ima-

geNet Challenge. Since Krizhevsky, Sutskever, and Hinton (2012) has used the ReLU function as an activation function in their ImageNet convolutional neural network, it has almost exclusively become the standard activation function for convolutional neural networks (Aggarwal (2018)). Therefore, the ReLU function should always be the first choice for the activation function.

Furthermore, a review of some of the most successful neural network architectures applied in ImageNet challenge, namely AlexNet (Krizhevsky, Sutskever, and Hinton (2012)), VGG (Simonyan and Zisserman (2014)) and ResNet (He, X. Zhang, Ren, and Sun (2016)), shows that the kernel size dimensions in the convolutional layers decrease from earlier to later layers in the network. Finally, the recommended values for the hyperparameters outlined in the previous Section 3.5.2 should also be considered for the design choices of convolutional neural networks.

To illustrate the exemplary architecture of a convolutional neural network, Figure 3.13 depicts LeNet, which has been one of the first successfully implemented convolutional neural networks. LeNet has been designed by LeCun, Bottou, Bengio, and Haffner (1998) and was applied for handwriting recognition. Instead of max-pooling, they have deployed average-pooling as the subsampling operation after the convolution operation.

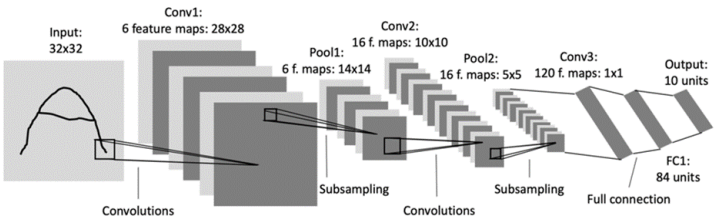

Figure 3.13: LeNet architecture.

3.6 Learning in neural networks

The supervised learning process, i.e. the class of machine learning algorithms used in this thesis, is briefly described in this section on the basis of a classification task. As described in Section 3.1.3, the dataset of a classification task consists of data points in the form (data:x,label:y), where each example or data point x_i has a target value y_i. Then, during the training phase of the neural

network, the prediction $\hat{y}_i$ for a data point x_i is compared to its true label y_i. Based on this, a so-called loss function $J(w)$ computes the error between the true labels y and the predicted values $\hat{y}$ as a function of the parameters of the neural network which are the values of its weights w. A very common loss function is the mean squared error defined in Equation 3.7 with n being the number of data points in the training dataset.

$$J(w) = \frac{1}{n} \sum_{i=0}^{n} (\hat{y}_i - y_i)^2 \tag{3.7}$$

The training process of a neural network is then a function minimization that adjusts the weights set w (including the biases) in such a way that the loss function is minimized (MacKay and Mac Kay (2003)). The learning of the correct weights starts from an initialization of the neuron weights at the beginning of the training process (Khan, Rahmani, Shah, and Bennamoun (2018)). Usually, the weights are initialized with values based on a certain distribution, like, for example, a Gaussian random initialization, while the biases are typically set to zero at the start of the training (Khan, Rahmani, Shah, and Bennamoun (2018)). When it comes to minimizing the loss function, every weight or bias represents an input parameter to the loss function and, hence, the minimization problem requires to calculate the gradient of the loss function with respect to its parameters w (MacKay and Mac Kay (2003)). Thus, neural networks are usually trained by using iterative gradient descent-based methods that drive the loss function to a low value by repeatedly adjusting the weights w. However, the calculation of the gradient of the loss function is non-trivial for feedforward networks because their loss is calculated from a complicated composition function over all weights of every neuron in ever neural network layer (Aggarwal (2018)). Thus, the backpropagation algorithm developed by Rumelhart, Hinton, and Williams (1986) has become a widely used algorithm that facilitates the calculation of the gradients. In the following sections, the basic procedure of the backpropagation algorithm and gradient descent methods for neural networks are briefly described.

3.6.1 Backpropagation

Explaining the details of the backpropagation algorithm would exceed the scope of this thesis, hence, only the basic concepts are introduced here according to

Aggarwal (2018). Further information can be found in Rumelhart, Hinton, and Williams (1986).

The backpropagation algorithm comprises two main phases which are referred to as the forward and backward phases, respectively. In the forward phase, the inputs of the data points from the training data are fed into the neural network. The inputs are then forward cascaded by executing the computations across the layers, using the current set of weights. The final predicted outputs which result from these computations are then compared to true labels y of the training data. Finally, the derivative of the loss function with respect to the predicted output is calculated. The main goal of the backward phase is then to learn the gradient of the loss function with respect to the different weights in the network. These gradients are then used to update the weights of the neural network. Since these gradients are learned in backward direction starting from the output node, this gradient learning process is called backward phase.

3.6.2 Gradient descent variants for neural networks

The backpropagation algorithm is often misunderstood as the whole learning algorithm for neural networks, even though it indeed only refers to the method of computing the gradient of the loss function (Goodfellow, Bengio, and Courville (2016)). The actual learning is performed by gradient descent while employing the gradient calculated by the backpropagation algorithm. Gradient descent belongs to the most popular algorithms for any kind of optimization task and is also the most common method to optimize neural networks (Ruder (2016)).

Gradient descent is a way to minimize the loss function $J(w)$ parametrized by the model's parameters $w \in \mathbb{R}^n$ by updating the parameters in the opposite direction of the gradient of the loss function $\nabla_w J(w)$ with respect to the parameters w. Thereby, the learning rate α determines the size of the steps that are taken to reach a (local) minimum (Ruder (2016)). The iterative character of gradient descent origins from adjusting the weights of the neural network by passing every single data point of the training dataset several times through the neural network. The cycle of passing through an entire dataset is referred to as epoch (Aggarwal (2018)). After a number of iterations when the parameters of the network do not change any longer as a result of the updates,

the network training process is said to converge (Khan, Rahmani, Shah, and Bennamoun (2018)).

Three different variants of gradient descent can be distinguished, namely batch gradient descent, stochastic gradient descent and mini-batch gradient descent. Each differs in how many of the data points of the training set are used to compute the gradient of the loss function. The mini-batch gradient descent tries to combine the advantages of the other two approaches by considering mini-batches of training data at each step of the weight updating process. As a result, it is generally the most common algorithm when it comes to neural network training, because its convergence behavior is stable and computing the gradient is quite efficient.

Further details about the three gradient descending methods can found in Ruder (2016).

3.7 Deep neural network-based decision tools

The idea of this thesis, i.e. the application of neural network-based reactive policies to the RCPSP, has been inspired by the recent success of deep learning in many challenging decision-making problems.

In robotics, deep neural networks have been widely used. One of the most common task in this field tackled with convolutional neural networks is object detection (Krizhevsky, Sutskever, and Hinton (2012)). This problem can also be seen as a decision-making task since the neural network must decide which object is in an image among a set of predefined objects, which the neural network have been trained to recognize. Another common robotic task which have been recently tackled with artificial neural networks applied to computer vision is the grasping of objects (Colling, Dziedzitz, Furmans, Hopfgarten, and Markert (2018)). Although the input information is still an image, the output of the neural network identifies how the gripper should grasp the object.

Kober, Bagnell, and Peters (2013) provides other examples for the field of robotics, which are not directly related the image processing such as, for instance, Hafner and Riedmiller (2003) and Riedmiller, Gabel, Hafner, and Lange (2009). In these works fully connected neural networks were used to let robots learn to play soccer and, precisely, to learn various sub-tasks such as defense, interception, position control, kicking, motor speed control, dribbling,

and penalty shots. Moreover, in Duan, Cui, and Yang (2008) and Thrun (1995) fuzzy and explanation-based neural networks have allowed robots to learn basic navigation, while CMAC (cerebellar model articulation controller) neural networks have been used for biped locomotion in Benbrahim and Franklin (1997).

Deep neural networks have been successfully applied to master different types of games as well. In this case, the decision-making process takes the current state of the game, i.e. the image displayed by the monitor at the decision point, as input of the neural network and decides what should be the next move. For example, Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, and Riedmiller (2013), Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, and Ostrovski (2015) and Hosu and Rebedea (2016) have applied this concept to the Atari games. Figure 3.14 taken from Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, and Ostrovski (2015) provides an example with the game Breakout, one of the Atari games, and its learning behavior. It is possible to see that in the earliest steps decision tool just try to hit the lowest levels of the bricks with a low rewarding, while later on it manages to break through the top level and a higher reward is given. With this reward mechanism, the neural network will learn after many runs that trying to break through the wall is the best strategy.

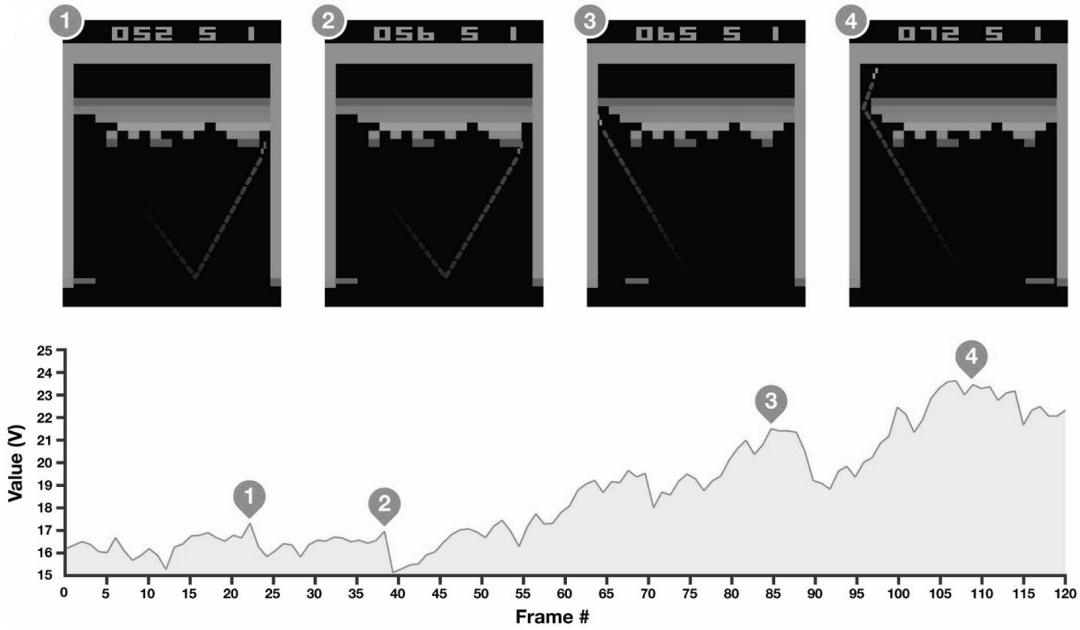

Figure 3.14: Visualization of learned value function on Breakout Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, and Ostrovski (2015).

A couple of years later, a much more difficult game called Go has been mastered Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, and Lanctot (2016) achieving a new important milestone in the world of artificial intelligence thanks to the complexity of the game and of the applied algorithm. The algorithm's creators proposed to combine a tree-search algorithm with two neural networks helping to both reduce the width and the depth of the search.

One of the first ideas for applying neural networks in the field of the production and logistics came from Leung (1995). However, in this thesis their potential of this field is only described from the theoretical point of view. Later on, Efendigil, Önüt, and Kahraman (2009) and Kochak and Sharma (2015) successfully applied neural networks for demand forecasting to support the supply chain management, while N. Silva, Ferreira, C. Silva, Magalhães, and Neto (2017) have investigated the problem of supply chain vulnerability and visibility with a similar approach by predicting the supply chain capacity to fulfill incoming orders and to anticipate which will be the next node receiving an order.

3.7.1 Application of DNN to resource-constrained problems

Resource management problems often require reactive decision making tools to take appropriate decisions based on the current state of the system. In this field of research, the contribution presented in Mao, Alizadeh, Menache, and Kandula (2016) is particularly relevant for this thesis, since it suggests the use of neural networks in a problem, which is quite similar to the resource-constrained project scheduling problem considered in this thesis. In both problems, a set activities requiring a certain amount of time and resources must be completed under the constraint of limited and renewable available resources. However, there are some major differences. In Mao, Alizadeh, Menache, and Kandula (2016), there is no known activity sequence with precedence constraints, instead new tasks randomly arrive at a waiting queue and the scheduler must decide which of them should be scheduled next. Moreover, the objective function (average job slowdown) is different and a reinforcement learning was used instead of supervised learning.

Although no contributions in the research field of the Resource-Constrained Project Scheduling Problem that tackled this problem with neural networks could be found, other machine-learning approaches have been already applied

to this problem. For what concerns the basic RCPSP (see Section 2.5.5), Jędrzejowicz and Ratajczak-Ropel (2014) and Adamu and Aromolaran (2018) have both applied reinforcement learning on the RCPSP but with different focuses. The first one used a multiagent system to improve the scheduling solutions, while the second one used machine learning to learn which algorithm should be dynamically chosen at each decision point among a set of heuristics. In both works, the trained machine learning model had no generalization purpose, i.e. the model was trained to solve only the activity sequence of the given dataset and it probably gives bad results on new unseen activity sequences. Moreover, both Jędrzejowicz and Ratajczak-Ropel (2014) and Adamu and Aromolaran (2018) assumed purely deterministic activity durations which is not very realistic as stated in 2.6.2.

To sum up, deep neural networks have been widely used for decision-making problems especially in the last decade. Thanks to the recent developments of convolutional neural networks, image processing is for sure one of their most important field of application, where they have achieved outstanding results. Moreover, they have been also tested in the other research fields, like, for example, the design of control policies and demand forecasting with good results. The RCPSP problem, however, is still new ground for deep neural networks, even if they have already been applied to similar problems like, for example, the job scheduling for computer clusters.

4 Applying deep neural networks to the RCPSP

Man's relation with technology is complex. We always invent technology, but then technology comes back and reinvents us.
-A. Jalan

This chapter introduces the author's idea of using deep neural networks to reactively schedule the activities of a project under precedence and resource constraints. The proposed approach has been tested and validated with a simulation tool which has been created in this thesis as well. Therefore, first of all, the simulation tool is presented. Then, the different decision-making policies are introduced by means of examples, including the six new neural network-based approaches. Finally, the entire training process used in this thesis is explained from the data generation with a so-called project generator, to the hyperparameter tuning and to the final test on the unseen test sequences.

The author's idea is based on a preliminary investigation and results presented in his previous publication on this topic (Pagani and Pfann (2020)).

4.1 Simulation tool

In order to evaluate one or more decision policies for an RCPSP problem, an evaluation tool is required, which is in the almost all related literature contributions a simulation tool (Vanhoucke (2016)). The simulation tool is used to build environments where an event-driven simulation can be run (Evans and Olson (2001)). If some characteristics of the simulation environment are stochastic (e.g. activity duration or resource consumption), the results are stochastic as well and, as a result, a statistical analysis (e.g. ANOVA ANalysis Of VAriance)

are required to draw solid conclusions. Few literature contributions have proposed exact evaluations. For example, in Rostami, Creemers, and Leus (2018) an evaluation based on Markov chains is used for small activity sequences with only 30 tasks, which is close the maximum number of activities exact evaluations can be applied for. Exact evaluations have another limit, namely that they are only available for a small set of probability distributions used for activity durations (e.g. deterministic, exponential or phase-type).

In this thesis, the simulation approach has been chosen due to its scalability when more than 30 activities are considered and in order to be able to model different distributions for the activity durations during the sensitivity analysis.

4.1.1 Simulation logics

Although the simulation logics refers to the formal definition of the basic resource-constrained project scheduling problem (see Section 2.6.1), a small example of simulation run is provided for a better understanding. The activity sequence of the example is depicted in Figure 4.1.

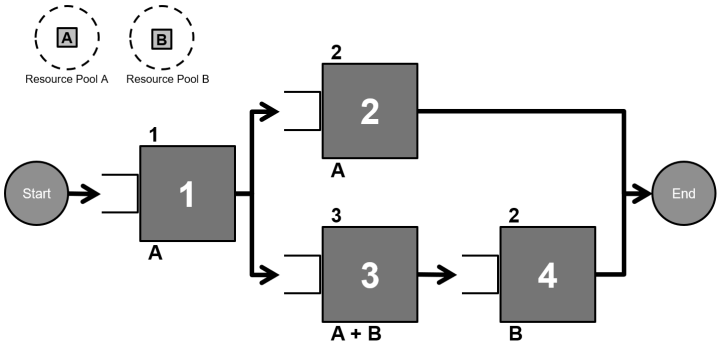

Figure 4.1: Considered example of activity sequence.

In particular, four activities are considered and each of them is depicted as a block with a number in the middle identifying the activity number. After the start dummy activity, activity 1 can be started. Once activity 1 has been completed, activity 2 and 3 could potentially start in parallel if enough resources were available. Once activity 3 has been completed, activity 4 can be started. Once all four activities are completed, the entire activity sequence declared as completed and the time span between its beginning and its completion is measured. According to Section 2.2.3, the duration of each activity is shown

directly over the block (e.g. activity 1 has a duration of 1 time unit), while the list of required resources below the block (e.g. activity 3 requires one piece of resource A and one of resource B). The resource pools are represented in the upper-left corner of Figure 4.1 as dashed circles where one piece of resource A and one of resource B are available and shared among all activities. In order to help the visualization, a waiting queue is added in front of each block.

The progress of the activity sequence during the simulation is shown by the so-called tokens. During the initialization of the simulation (see Figure 4.2), they are placed in all the blocks after the start dummy activity.

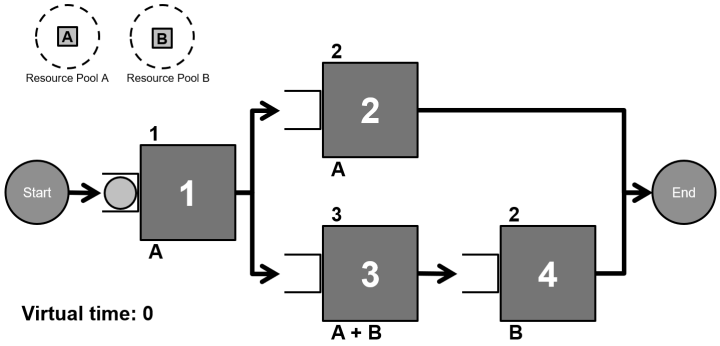

Figure 4.2: Initialization phase in the considered activity sequence.

Whenever an activity is completed, the token is duplicated (if there is more than one following activity), placed in the waiting queues of all following blocks and, if another one is already in one of those queues, they are merged into one token. Whenever an activity is started, the token is moved from the waiting queue to the middle of the block. As a result, the tokens show which activities are currently "in progress", "ready to start", "idle" or "not active". An activity is said to be "in progress" if it has been started and, as a result, a token is in the middle of the correspondent block. If a token is in the waiting queue, the activity is "idle" and that means that at least one predecessor has been completed but the activity itself has not been started yet. If the activity is "idle" and all previous activities have been completed the activity is also "ready to start". In all other cases, the activity is "not active" and there is no token in the block or in its waiting queue. That happens when no previous activities have been completed yet ("future" activity) or if the activity itself has already been completed ("past" activity).

The simulation is built with a 3-step logic and the three steps are iteratively executed until all activities have been completed. The three steps are the following ones:

- **Step 1 (decision step).** In the decision step, it is decided which activities can immediately start.
- **Step 2 (progress step).** The next finishing activity (or activities) in the event list is identified and the simulation jumps forward in time to that event.
- **Step 3 (event step).** The consequences of that event (e.g. the update of the token position and the correspondent splits and merges explained in the previous paragraph) are evaluated and the system state is updated accordingly.

The following figures show the system representation after each step in a step-by-step execution for the considered small example throughout an entire simulation run. After the initialization phase (Figure 4.2), the first iteration begins. At step 1 of iteration 1 (see Figure 4.3), the first decision takes place. Since only activity 1 can be started and there are enough resources available, the decision is trivial. Activity 1 is started and the resource A is allocated to it.

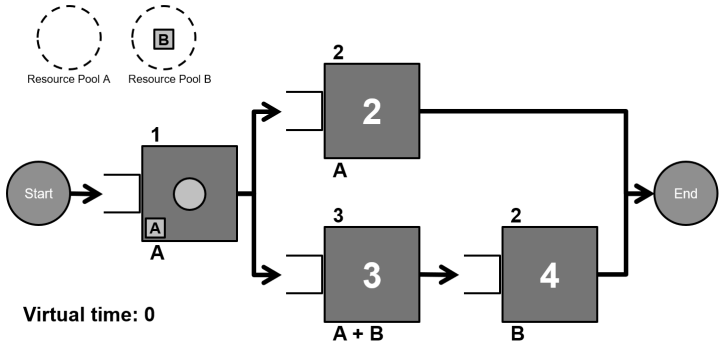

Figure 4.3: Step 1 of iteration 1 in the considered activity sequence.

The step 2 of iteration 1 (Figure 4.4) is the progress step. The next finishing activity is identified, the simulation jumps forward in time to the completion event and the virtual time is updated. In this case, activity 1 completes and, since its execution time is one time unit, the virtual time is increased by one.

Afterwards, there is step 3 (see Figure 4.5) where the completion event is evaluated. In this case, the token is split in two and a token is placed in each

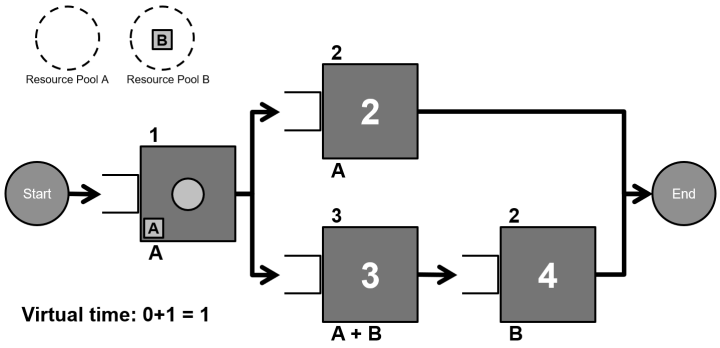

Figure 4.4: Step 2 of iteration 1 in the considered activity sequence.

waiting queue of both activity 2 and 3. The resources allocated to activity 1 are given back into the pool.

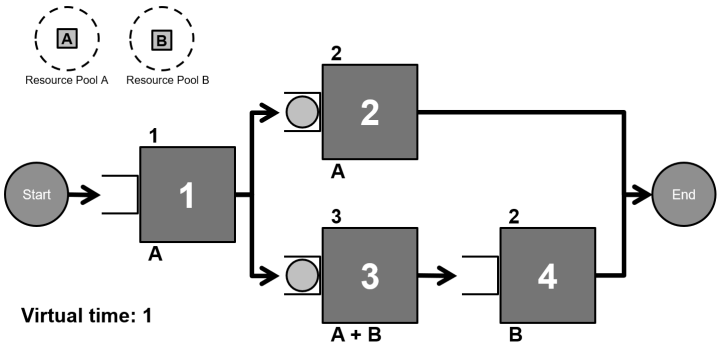

Figure 4.5: Step 3 of iteration 1 in the considered activity sequence.

Immediately after step 3 of iteration 1, the step 1 of the next iteration the next iteration takes place. In this decision step, both activity 2 and 3 could be potentially started, if enough resources would be available for both. However, since both needs a resource A and there is only one piece of it available, a non-trivial decision must be taken. Let us assume that it is decided to start activity 2 and the correspondent resource is allocated as depicted in Figure 4.6.

After step 1 of iteration 2, there are no more non-trivial decisions and the activities are executed one after each other in this sequence 1, 2, 3 and 4 without parallelizations (see Figures from 4.7 to 4.14). A total project duration of 8 time units is obtained in this case.

If a different decision was taken at step 1 of iteration 2, the following steps would have gone differently as shown in Figures from 4.15 to 4.20 and a total duration of the activity sequence of 6 time units instead of 8 would have been achieved.

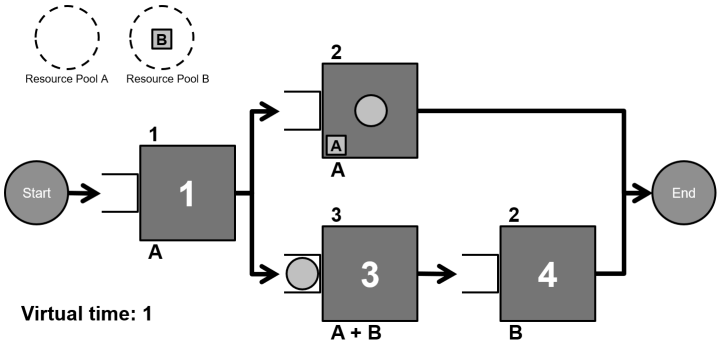

Figure 4.6: Step 1 of iteration 2 in the considered activity sequence.

This example shows that a good decision policy can be important even for small activity sequences since a reduction of 25% on the total duration has been achieved in this case.

4.2 Decision-making process with a reactive policy

As mentioned in Section 2.4, the scheduling of a project consists in determining when the activities should be started. In contrast to a proactive scheduling policy where the starting time of activities is decided prior to the beginning of the project, a reactive scheduling policy only defines which "ready to start" activities are actually started at each decision point. The decision points are in this case the beginning of the project (step 1 of the first iteration) and whenever an activity completes and, as a result, new resources are released. In the following sections, it is described which pieces of information can be used to generate the priority values for the "ready to start" activities and how the priority values are used to take the decisions. Finally, the considered policies for the benchmark in chapter 5 are outlined.

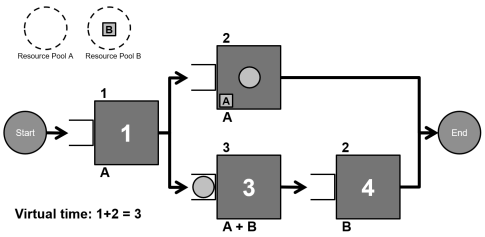

Figure 4.7: Step 2 of iteration 2

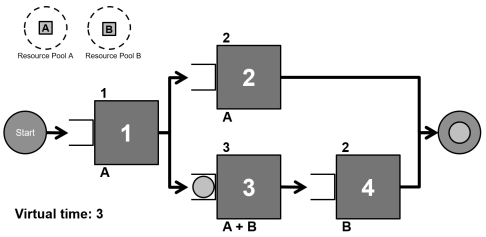

Figure 4.8: Step 3 of iteration 2

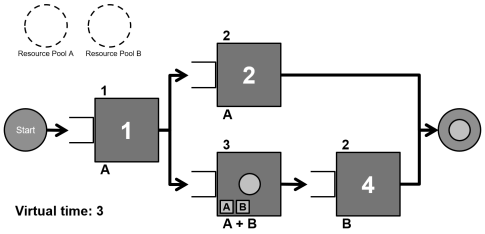

Figure 4.9: Step 1 of iteration 3

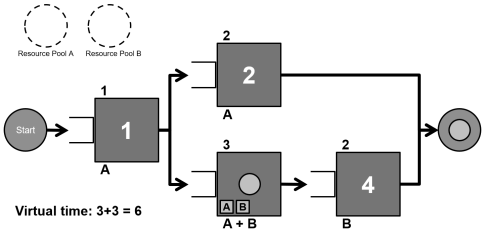

Figure 4.10: Step 2 of iteration 3

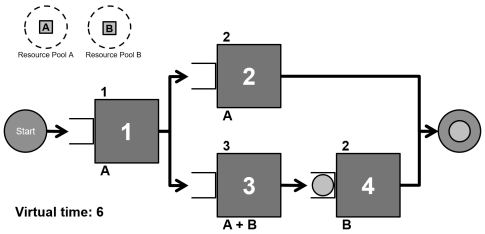

Figure 4.11: Step 3 of iteration 3

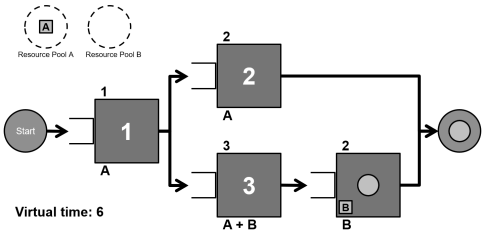

Figure 4.12: Step 1 of iteration 4

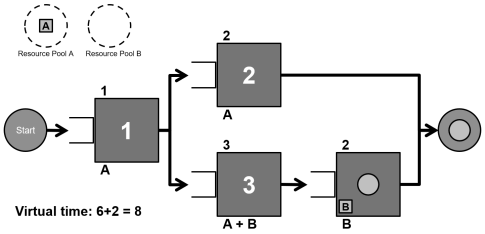

Figure 4.13: Step 2 of iteration 4

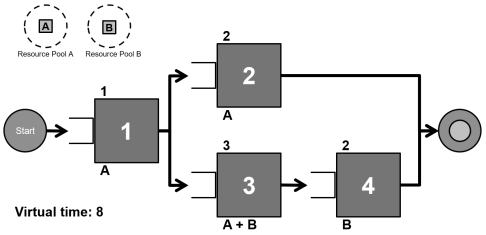

Figure 4.14: Step 3 of iteration 4

4.2.1 Available information

The first step of designing decision policies is to identify which pieces of information could be considered in the decision-making process at each decision point t_d. In this thesis, seven categories of information have been considered:

- The "idle" activities (set $\mathbb{I}$), i.e. the activities with a token in the waiting queue, which means that at least one previous activity has been completed but they have not been started yet.

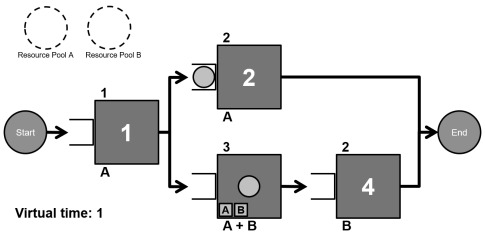

Figure 4.15: Step 1 of iteration 2

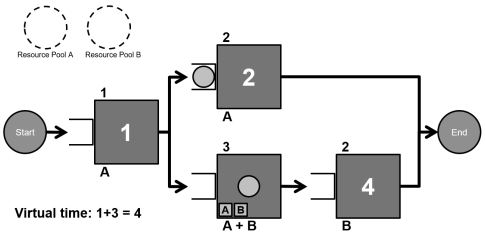

Figure 4.16: Step 2 of iteration 2

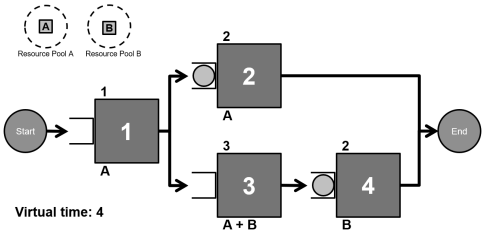

Figure 4.17: Step 3 of iteration 2

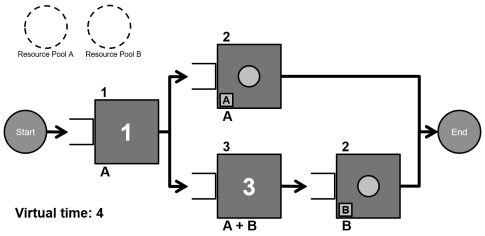

Figure 4.18: Step 1 of iteration 3

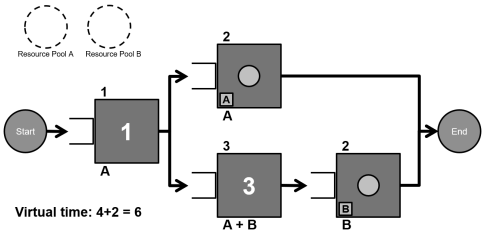

Figure 4.19: Step 2 of iteration 3

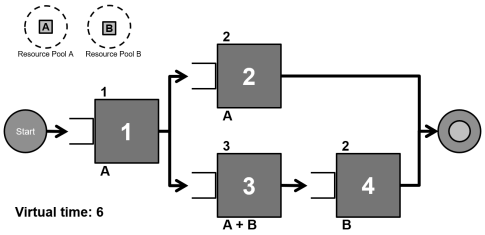

Figure 4.20: Step 3 of iteration 3

- The "ready to start" activities (set $\mathbb{R}$), i.e. the activities that could be scheduled standalone at the decision point t_d and, as a result, are considered in the decision. They correspond to all "idle" activities where all precedence and resource constraints are satisfied at $t = t_d$.
- The "in progress" activities (set $\mathbb{P}$), i.e. the activities that have been started at $t < t_d$ but they are still using resources at $t = t_d$.
- The "future" activities (set $\mathbb{F}$), i.e. the activities that have not been started yet and cannot be scheduled at t_d due to precedence or resource constraints. They correspond to all "idle" activities, which are not "ready to start" at $t = t_d$.
- The precedence constraints among these four types of activities.
- Currently available resources.
- Total number of resources.

For each of the considered activities, it is possible to use different pieces of information, namely the duration and the resource consumption for each type of resource.

It is important to notice that the already completed activities are not considered anymore. In this thesis, it is assumed that those activities have no influence on the current decision.

4.2.2 From the priority values to decision

The goal of a reactive scheduling policy is to convert the available information into priority values for the "ready to start" activities and, then, to convert the priority values in a list of "ready to start" activities that are actually started. In order to do that, the "ready to start" activities are considered one after each other starting with the activities with the highest priority value. If there are enough resources to schedule the currently considered activity, it is scheduled at t_d and the correspondent resources are allocated. After that, the same procedure is applied to the next activities one after each other considering the order given by the priority values. The algorithm should not stop as soon as a "ready to start" activity cannot be scheduled, since it may happens that the next one has a lower resource consumption and therefore can be scheduled. This algorithm is greedy, namely that the possibility not to schedule a "ready to start" activity although there are enough resources is not considered.

4.2.3 Considered policies

In order to evaluate the performance of the neural network-based scheduling policies proposed in this paper, a set of other policies has been considered for the performance comparison. Those policies are the random policy, the lower bound policy and the heuristics presented in Section 2.9.1. The following list provides all the scheduling policies along with the description of their scheduling algorithm that have been tested in this thesis:

- **Random (RAN).** The priority values are assigned randomly. Due to the random nature of this policy, the variance of a project makespan may be significantly larger than for other scheduling algorithms. In order to significantly reduce variance, 10.000 simulation runs are considered and the average makespan is used for benchmarking purposes (see section 4.4).

- **Lower bound with parameter N (LB-N).** The policy performs N simulations with a random policy from the considered decision point till the project completion and, considering the run with the lowest total duration, the activities scheduled to start at the decision point are started. In this thesis, the default value of N has been set to 10.000.
- **Shortest Imminent Operation (SIO).** The lower the duration of a "ready to start" activity, the higher its priority.
- **Greatest Total Resource Demand (GTRD).** It schedules the activities on the basis of the total resource demand, namely that the higher the resource demand, the higher its priority value. Critical resources are given a greater importance which means that an activity with a high resource demand of a critical resource is given a bigger priority than one with a high resource demand of another resource. For more details, see Section 4.2.3.
- **Greatest resource utilization with parameters S and T (GRU-S-T).** The combination of the starting activities are chosen in a way that the resource utilization in a time horizon of T time units is maximized. In particular, S schedules complying with the resource and precedence constraints and considering a time horizon of T time units are generated, the correspondent resource utilizations are computed and the activities scheduled at the current decision point t_d in the schedule with the highest resource utilization are given the highest priority values. For more details, see Section 4.2.3.
- **Fully connected neural network without future resource utilization (NN-FC).** Explained in Section 4.3.3.
- **Fully connected neural network with future resource utilization (NN-FC-FRU).** Explained in Section 4.3.4.
- **Convolutional 1-dimensional neural network without future resource utilization (NN-CONV1D).** Explained in Section 4.3.5.
- **Convolutional 1-dimensional neural network with future resource utilization (NN-CONV1D-FRU).** Explained in Section 4.3.6.
- **Convolutional 2-dimensional neural network without future resource utilization (NN-CONV2D).** Explained in Section 4.3.7.
- **Convolutional 2-dimensional neural network with future resource utilization (NN-CONV2D-FRU).** Explained in Section 4.3.8.

The assumption of this thesis is that, although the activity durations are modeled as known and deterministic, some deviations from the assumed values can occur. As a result, it is expected that the further in the future an activity is scheduled, the more likely is that its starting time becomes unfeasible. For this reason, a limited time horizon is considered in all scheduling policies that consider also the future activities, i.e. the GRU-S-T and the neural network approaches.

Greatest Total Resource Demand Policy

This section provides a formal description of how the greatest total resource demand policy (GTRD), has been implemented in the simulation tool used for the benchmark. The algorithm uses in the following steps:

- **Step 1: compute the resource criticality values.** The criticality of a resource type indicates how limiting a certain resource type is among the "ready to start" activities. If few pieces of that resource type are available and many "ready to start" activities require many units of it, the resource is very limiting. The formula for the resource criticality value for the resource type k, denoted as RCV_k, is presented in equation 4.1.
- **Step 2: compute the activity criticality values.** For each activity, a criticality value, which is proportional to the consumption of the most critical resource type, can be computed. If two or more activities get the same priority value, the second most critical resource type is considered to define the relative position among them and so on.
- **Step 3: assign priority values.** A priority value PV_j which is proportional to the activity criticality value is assigned to each activity.

$$RCV_k = \frac{(\sum_{i \in \mathbb{R}_d} r_{j,k}) - A_k}{R_k} \tag{4.1}$$

Considering the second decision point of the considered example of Section 4.1.1 (Figure 4.21), the following computations have to be done:

$$\begin{cases} RCV_A = \frac{(1+1)-1}{1} = 1 \\ RCV_B = \frac{(0+1)-1}{1} = 0 \end{cases} \tag{4.2}$$

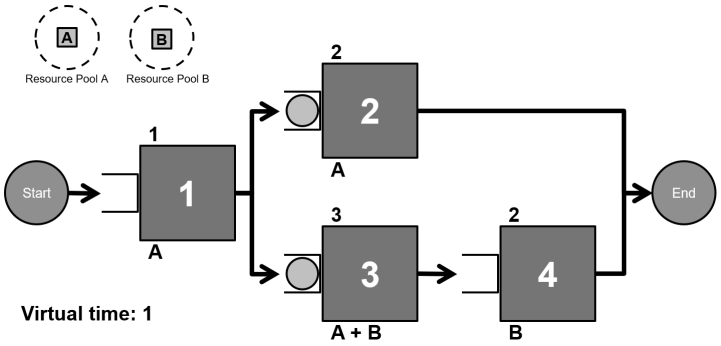

Figure 4.21: Second decision point for the considered example.

The resource type A results to be the most critical resource and the following priority values can be computed:

$$\begin{cases} PV_2 = 1 \cdot r_{2,A} = 1 \\ PV_3 = 1 \cdot r_{3,A} = 0 \end{cases} \tag{4.3}$$

In this case, the two "ready to start" activities have different priority values. If that was not be the case, the same computation should be repeated with the second most critical resource type, i.e. with B.

Greatest Resource Utilization Policy

This section aims to provide a rigorous description of how the greatest resource utilization policy (GRU), has been implemented in the simulation tool used for the benchmark. This policy relies on two parameters, namely the number of generated schedules S and the length of the time horizon T in which the schedules are generated. Consequently, many different sub-classes of this policy denoted as GRU-S-T can be considered. The schedules must also comply with the resource and precedence constraints and are created with the following steps:

- **Step 1: initialize the schedule.** An empty schedule for the time horizon T, which is a resource allocation diagram with information about the start and end time of the scheduled activities along with their resource consumption, is created (Figure 4.22) and the "in progress" activities are added from t_d until the planned end time.
- **Step 2: add next activities.** For each schedule created up to now, it is evaluated if further activities can be placed in the schedule with $t_d \leq s_j \leq$

$t_d + T - 1$ and considering the precedence and resource constraints. s_j is the activity's start time. If different combinations of further activities can be placed, child schedules are generated and the parent schedule is deleted since its resource utilization will be always lower than the one of a child schedule according to equation 4.4. In the considered example, 2 child schedules are generate at the first iteration, namely the ones depicted in Figures 4.23 and 4.25.

- **Step 3: check stop criteria.** A full schedule is a schedule where no more activities can be placed within the time horizon T. If the desired number of full schedules S is reached or no more new child schedules can be generated, the schedule generation algorithm stops, otherwise, step 2 is repeated.

For each obtained full schedule, the resource utilization is computed according to equation 4.4 and the "ready to start" activities scheduled at the current decision point t_d in the schedule with the highest resource utilization get a priority value of one, while the others get a priority value of zero.

Equation 4.4 shows how to compute the resource utilization RU_s of a schedule s in the time horizon T. $A_{k,t}$ is the current available quantity of units for the k^{th} resource type at time t.

$$RU_s = \frac{\sum_{k=1}^{K} \sum_{t=t_d}^{t_d+T-1} \frac{R_k - A_{k,t}}{R_k}}{K \cdot T} \tag{4.4}$$

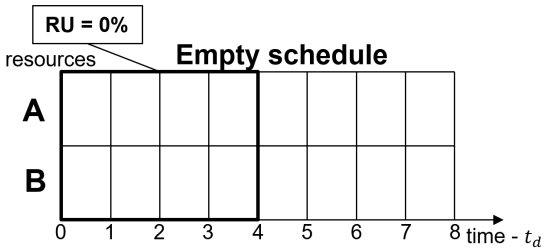

Figure 4.22: Empty schedule

In order to better understand the policy, the generated schedules for the second decision point of the considered example of Section 4.1.1 (Figure 4.21) are presented. The example is valid for $T = 4$ and $S \geq 2$ since no more than two schedules can be generated for this small example. Figures 4.24 and 4.26 show the final full schedules. Schedule 1 has a resource utilization of 75%, schedule

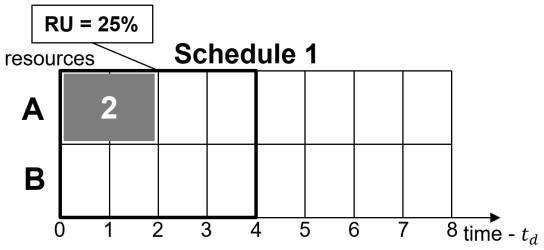

Figure 4.23: Schedule 1 at the first iteration

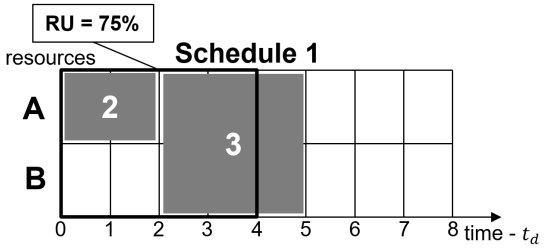

Figure 4.24: Schedule 1 at the second and last iteration (full schedule)

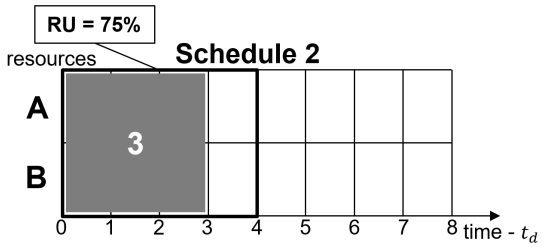

Figure 4.25: Schedule 2 at the first iteration

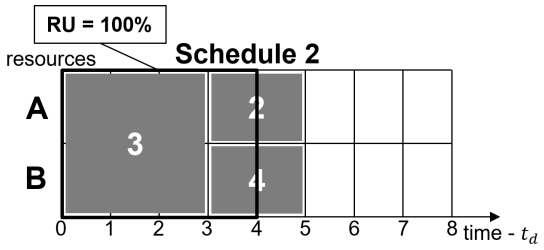

Figure 4.26: Schedule 2 at the second and last iteration (full schedule)

2 uses 100% of both resources for the entire time horizon. As a result, the GRU scheduling policy with $S \geq 2$ and T=4 suggests to start activity 3 at the considered decision point.

4.3 Deep neural network structure

As mentioned in Section 3.7, a neural network can be use as a decision tool in many different problems. The idea of this thesis is that such powerful computing systems can be applied to the resource-constrained project scheduling problem as well. This section presents the methodology that makes it possible both with a formal explanation and with the simple but representative example of the decision point in Figure 4.27.

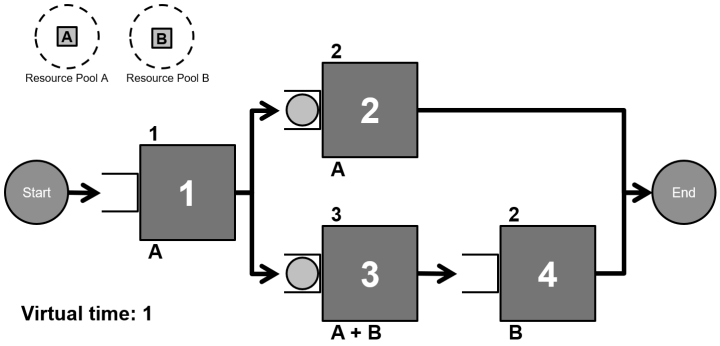

Figure 4.27: Second decision point for the considered example.

Six different neural network structures are tested in this thesis. Three of them consider only the information about the "ready to start" activities and the currently available resources. Their architecture can be represented by Figure 4.28. The block "Input 1", also denoted as $V_{ReadyToStartActivities}$, is the considered input information and it is processed by the first neural network, denoted as NN_1, which can be a fully connected, a convolutional 1-dimensional or a convolutional 2-dimensional one. After this neural network, the processed information is flattened and given as an input to the so-called NN_{final}. The latter is a fully connected neural network which trasforms this intermediate processed information into the output values of the entire network, i.e. the priority values.

The other three neural network structures consider also the information about the future resource utilization and their architecture can be represented by Figure 4.29. The additional piece of information is denoted as "Input 2" or $M_{FutureResourceUtilization}$ and is initially processed by a convolutional 2-dimensional neural network, denoted as NN_2. After that, the information is flattened and merged with the information coming out from NN_1. The merged information is than processed by the NN_{final} to obtain the output values of the entire network, i.e. the priority values.

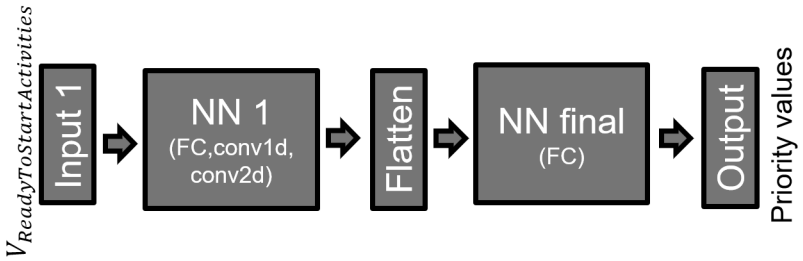

Figure 4.28: Neural network structure without considering the future resource utilization.

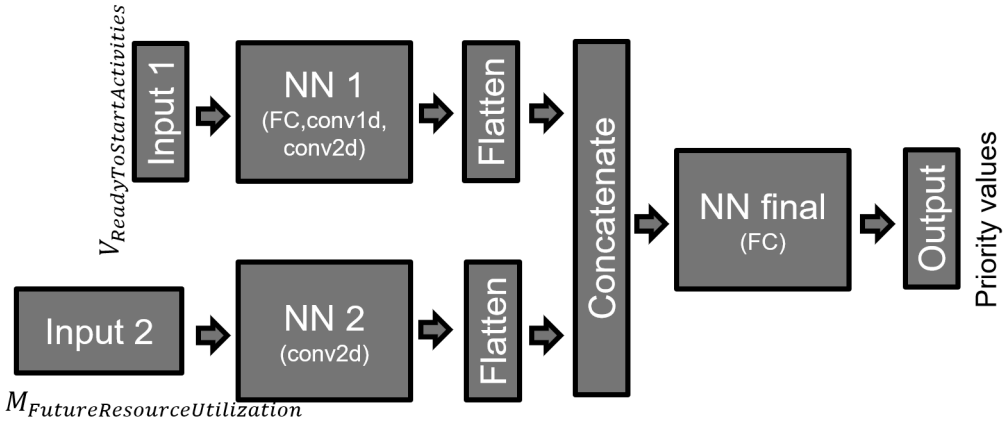

Figure 4.29: Neural network structure considering the future resource utilization.

4.3.1 Input structure

The two pieces of input information, i.e. the ready to start activity state vector $V_{ReadyToStartActivities}$ (input 1) and the future resource utilization matrix $M_{FutureResourceUtilization}$ (input 2), are created according to the rules explained in this section. One of the biggest challenges in using a deep neural network to process this information is that the number of "ready to start" activities and the number of following activities generally vary at each decision point, while the number of inputs of a neural network remains fixed. A similar problem was faced also by Mao, Alizadeh, Menache, and Kandula (2016), where neural networks are used for the allocation of computational resources of a CPU cluster to a number of pending jobs. Since the jobs arrive with a random inter-arrival time, the waiting queue can grow and no maximum queue size can be defined. The author proposed to consider the full information only about a limited and predefined amount of waiting jobs, while, for what concerns the other

pending jobs, their number is the only information being given as an input and considered in the decision process.

Ready to state activity vector

In order to solve the problem of the variable number of "ready to start" activities, a maximum number of considered ones, denoted as R_{max}, is considered. If more activities are ready to start, the information of the activities in excess is not processed and random low priority values are assigned in such a way that they are considered at last in the decision process (see Section 4.2). If less activities are ready to start, the correspondent positions of the state vector are set to zero.

Mao, Alizadeh, Menache, and Kandula (2016) also considered limited number of activities in state vector

Assuming that R_{max}=3 "ready to start" activities (1, 2 and 3) are considered at maximum and K=2 resource types (A and B), the ready to start activity state vector $V_{ready_to_start_activities}$ becomes an vector with $(1+K) \cdot R + K$ positions with the following structure:

$$V_{ReadyToStartActivities} = \begin{bmatrix} d_1 \cdot RF \\ \frac{r_{1,A}}{R_A} \\ \frac{r_{1,B}}{R_B} \\ d_2 \cdot RF \\ \frac{r_{2,A}}{R_A} \\ \frac{r_{2,B}}{R_B} \\ d_3 \cdot RF \\ \frac{r_{3,A}}{R_A} \\ \frac{r_{3,B}}{R_B} \\ \frac{A_A}{R_A} \\ \frac{A_B}{R_B} \end{bmatrix} \tag{4.5}$$

Excluding the last K values, the vector can be divided in groups of $1 + K$ values and each group refers one of the "ready to start" activities. The first value of each group represents the duration, while the following ones represent the normalized resource utilization $NRU_{j,k}$, i.e. the resource consumption in percentage with respect to the total available quantity (see Equation 4.6). RF is a rescale factor for the activity duration that normalizes the correspondent input

values in such a way that they are between zero and one, which is a common best practice in neural network models called feature scaling. According to Ketkar and S. (2017), this practice improves the convergence of the gradient descent and therefore leverages the learning process of the neural network. The last K values represent the currently available quantity of units for each resource type.

$$NRU_{j,k} = \frac{r_{j,k}}{R_k} \tag{4.6}$$

Considering R_{max}=3, K=2 and RF=0.1, the $V_{ReadyToStartActivities}$ for the second decision point of the considered example depicted in Figure 4.27 is as follows:

$$V_{ReadyToStartActivities} = \begin{bmatrix} 0.2 \\ 1 \\ 0 \\ 0.3 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{4.7}$$

It is important to notice that activity 2 is the first "ready to start" activity according to the activity numbering and, as a consequence, it gets the index 1. Furthermore, activity 3 gets the index 2 and since there are only two "ready to start" activities, the vector positions for the third activity index are set to zero.

Future resource utilization matrix

The second piece of information received by the neural network as an input is the future resource utilization in form of a $[K\text{x}T]$ matrix, where K is the number of resource types and T is the considered time horizon. A generic element $M_{k,t}$ of this matrix represents a rough forecast about how much the resource type k is required t time units after the decision point t_d. In order to fill this matrix, a fictitious schedule without considering the resource constraints is created, which is equivalent to a schedule made using the LB-CPM scheduling policy (see

Section 2.8). The fact that the resource constraints are not considered makes this schedule unique, i.e. there is no other alternative schedule, if all activities are placed in the schedule as soon as all previous activities are completed. Such a fictitious schedule is created with the following steps:

- **Step 1: initialize the schedule.** An empty schedule for the time horizon T is created and the "in progress" activities are added from t_d till the end of the considered time horizon. Figure 4.22 provides an example for $T = 4$ and $K = 2$.
- **Step 2: add the next activities to the schedule.** It is evaluated if further activities can be placed in the schedule with $t_d \leq s_j \leq t_d + T$, with s_j being the activity's start time. The resource constraints are not considered.
- **Step 3: check stop criteria.** If no more activities can be placed within the time horizon T, the schedule generation algorithm stops, otherwise, it goes back to step 2.

Once the fictitious schedule has been obtained, an empty resource utilization matrix with K rows and T columns is created and filled. The following algorithm is used for each scheduled activity j and for each resource type k.

- **Step 1: compute the normalized resource utilization.** The normalized resource utilization $M_{j,k}$ of the considered scheduled activity j and for the resource type k is calculated according to equation 4.6.
- **Step 2: add the normalized resource utilization in the resource utilization matrix.** The values of the future resource utilization matrix of the row k and of the columns between $max(0, s_j - t_d)$ and $min(T - 1, s_j - t_d + d_j)$, i.e. in the time slots of the considered time horizon where the activity has been scheduled, are incremented by $M_{j,k}$.

In order to better understand how the future resource utilization matrix is created, the algorithm is applied to the second decision point of the considered example depicted in Figure 4.27.

In the first part of the algorithm the fictitious schedule is created. At the decision point $t_d = 1$, activities 2 and 3 could be scheduled immediately ($s_2 = 1$ and $s_3 = 1$) if the resource constraints were not considered, while activity 4 must wait (only) for the completion of activity 3 ($s_4 = s_3 + d_3 = 4$). The resulting schedule is depicted in Figure 4.30.

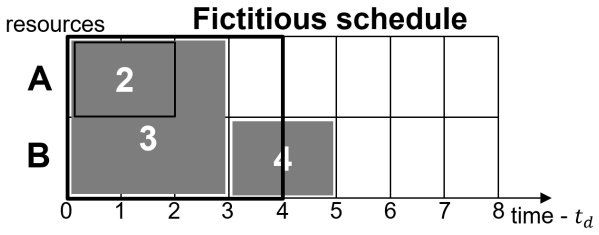

Figure 4.30: Fictitious schedule.

In the second part of the algorithm, the future resource utilization matrix is created starting from an empty one shown in Equation 4.8. Three activities have been scheduled within the time horizon T from the decision point t_d, namely activity 2, 3 and 4. Considering the first row dedicated to the resource type A and the second one to the resource type B, the Equations 4.9, 4.10 and 4.11 represent the future resource utilization matrices for the activity 2, 3 and 4 respectively. If these three matrices are added element by element, the matrix in equation 4.12 is obtained, which represents the input for the neural network NN_2.

$$M_{FutureResourceUtilization,empty} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.8}$$

$$M_{FutureResourceUtilization,Activity2} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.9}$$

$$M_{FutureResourceUtilization,Activity3} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \tag{4.10}$$

$$M_{FutureResourceUtilization,Activity4} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.11}$$

$$M_{FutureResourceUtilization} = \begin{bmatrix} 2 & 2 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{4.12}$$

By creating the future resource utilization matrix, the information outlined in Section 4.2.1 about what comes after the "ready to start" activities is compressed into a matrix of a predefined shape. As a result, part of the information can go lost, which can be easily proved with the help of Figure 4.31. The first situation is the one presented in Figure 4.27. The second situation includes the

same activities as in the first one but the topology is slightly different. With the second topology the possibility to parallelize some activities at some point disappears and, as a result, a total duration of 8 time units is always obtained no matter if activity 3 is done before activity 2 or viceversa. That means that the assigned priority values have no effect on the total project duration.

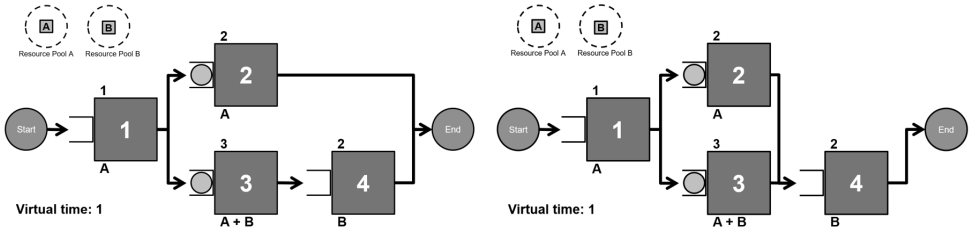

Figure 4.31: Comparison between two different situations that have the same future resource utilization matrix.

Although an activity scheduler should recognize this difference, it can be easily computed that both situations have the same input information, i.e. both the same vector $V_{ReadyToStartActivities}$ (Input 1) and matrix $M_{FutureResourceUtilization}$ (Input 2), with the proposed approach. This problem leads of course to a certain degradation of the performance of the proposed neural-network-based scheduling policy. However, this degradation is accepted as a compromise to have an input information of a predefined and fixed shape which is suitable for the processing with a neural network.

Resource conversion

In order to improve the performance of a trained neural network as a decision tool, one should make sure that similar situations requiring the same decision are expressed with similar input information. In Figure 4.32 two situations are presented which only differ from each other in the resource consumption. In the second situation, the resource consumptions for the resource A and B have been switched but the decision-making process does not change and activity 3 should still get a higher priority value than activity 2 to achieve a lower total project duration.

As a result, a resource conversion has been proposed in this thesis and it consists in changing the considered order of the resource types according to the current resource criticality RCV_k defined in 4.1. In particular, the resource types with a higher criticality are considered first and that has an effect on the order of

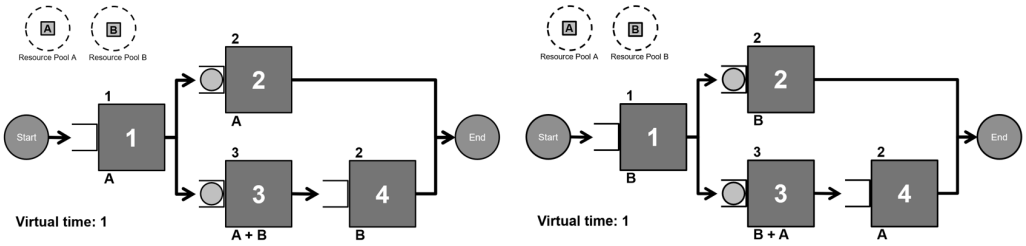

Figure 4.32: Comparison between two similar situations where the resource consumption of resource A and B are inverted.

the elements of the input information, i.e. both the vector $V_{ReadyToStartActivities}$ and the matrix $M_{FutureResourceUtilization}$.

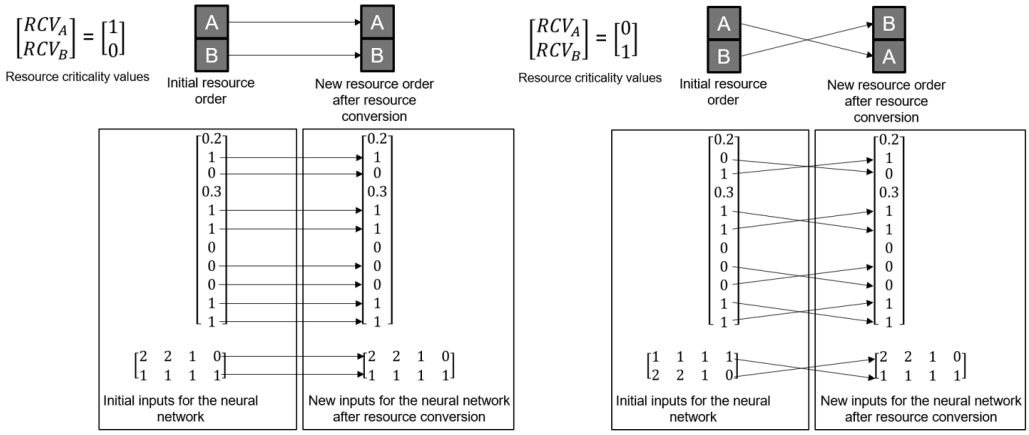

Figure 4.33: Modification of the input information in both situations after the resource conversion.

Figure 4.33 shows how the input information changes in both considered situations. Since they are different, the initial input vector and matrix are different. However, after the resource conversion the same input is achieved and, as a result, also the same priority values, which is the goal of the resource conversion. The usefulness of the resource conversion in the considered class of problems is tested and confirmed later with post-validation experiments in Section 5.5.5.

Activity conversion

As mentioned in the previous sections, the neural network can consider a maximum number of R_{max} "ready to start" activities in the vector $V_{ReadyToStartActivities}$ which according to the input notation are numbered from 1 to R_{max}. This

is called local numbering since it considers only the activities which could be started at the decision point t_d. However, the "ready to start" activities have in general a different numbering in the activity sequence, denoted as global numbering. As a result, an activity conversion rule must be defined to relate the global to the local numbering and to define which information goes in which position of the input vector and matrix.

The solution to this problem proposed in this thesis is very similar to the computation of the activity priority values in the GTRD scheduling policy. The only different is that the obtained activity criticality values ACV_j (see Section 4.2.3), which are proportional to the consumption of the most critical resource type, are used to order the activities in ascending order. As for the GTRD scheduling policy, if two activities get the same value because they have the same consumption of the most critical resource type, the second most critical resource type is considered for their relative position and so on.

This methodology helps to solve problems like the one presented in Figure 4.34 where two situations are presented which only differ from each other in the activity numbering. In the second situation, activity 2 has become activity 3 and vice versa. The two situations represent basically the same one except for the different activity numbering. As a result of the different relation between local and global numbering, different input information are obtained and probably a different output of neural network which may lead to a different decision. Considering a sorting rule which is independent from the global numbering is an effective way to tackle this problem as proven in Section 5.5.5.

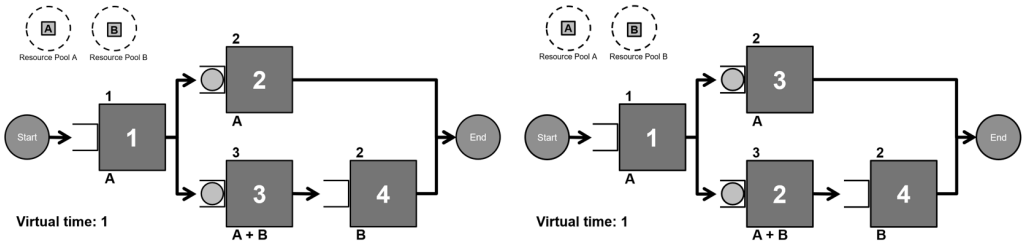

Figure 4.34: Comparison between two different situations where activity 2 has become activity 3 and vice versa.

The first step to order the "ready to start" activities is to define which resource is the most critical. Equation 4.1 can be applied and the following results are obtained:

$$\begin{cases} RCV_A = \frac{(1+1)-1}{1} = 1 \\ RCV_B = \frac{(0+1)-1}{1} = 0 \end{cases} \tag{4.13}$$

The resource type A is the most critical. However, since all "ready to start" activities have the same resource consumption for A, the second most critical resource type, i.e. B, is considered. In both cases and independently from the global numbering, the "ready to start" activity of the upper block gets the local index 1, while the one of the lower block gets the local index 2 and this leads to the same input information as shown in Figures 4.35, which is the goal of the activity conversion.

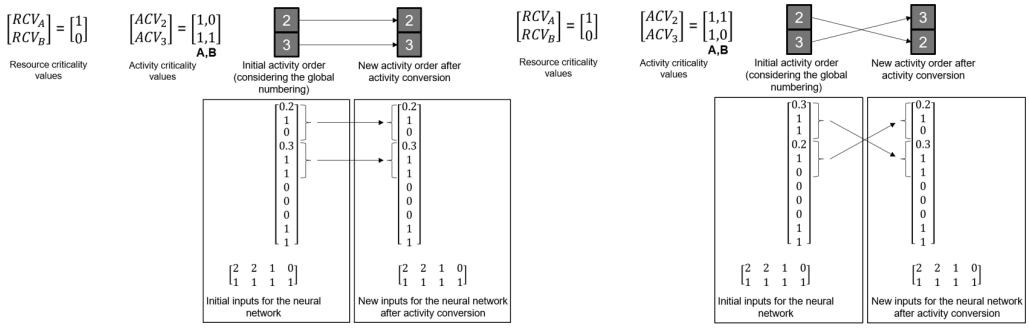

Figure 4.35: Modification of the input information in both situations after the activity conversion.

The usefulness of the activity conversion is tested and confirmed in Section 5.5.5.

4.3.2 Output structure and decision process

Once the input information has been processed through the neural network, an output vector with a number of values equal to the maximum number of considered "ready to start" activities R_{max} is obtained. This vector contains the priority values that will be assigned to the "ready to start" activities. The correspondence between these activities and the values of the vector is given by their order after the activity conversion (see Section 4.3.1).

For the considered example introduced in Figure 4.27 and after the resource and activity conversion, the input and output structure is depicted in Figure 4.36. As the reader may notice, the output values, which are just by an example, refers to the set of "ready to start" activities with the same order as for the input vector $V_{ReadyToStartActivities}$.

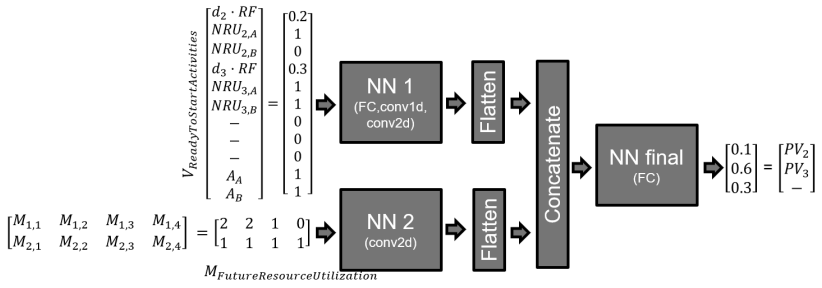

Figure 4.36: Representation of the input and output information for the considered example.

The decision process takes place just after the creation of the priority output vector and consists in trying to start the considered "ready to start" activities one after each other starting with the activities with the highest priority value. In the example depicted in Figure 4.36, activity 2 and 3 are given the priority values of 0.1 and 0.6 respectively. As a result, the algorithm tries to start activity 3 at first. Since enough resources are available, it is successfully started. Afterwards, the possibility to start of activity 2 is evaluated without success since all resources have been allocated to activity 3. At this point, the step 1 of iteration 2 is completed and the simulation can proceed with step 2.

4.3.3 Fully connected neural network without future resource utilization

As already mentioned in Section 4.2.3, different neural networks are proposed as decision tools in this thesis. The first one is called fully connected neural network without future resource utilization, also denoted as NN-FC, and its structure along with the considered hyperparameters are shown in Figure 4.37.

The first part of the name, in this case "fully connected" always refers to the neural network type of NN_1. In the NN-FC neural network both NN_1 and NN_{final} are fully connected and there is no information flattening or merging between them. As a result, it is possible to consider them as a unique fully connected neural network.

The considered hyperparameters for this neural network structure are:

- The maximum number of considered "ready to start" activities R_{max}.
- The total number of fully connected layers in NN_1 and NN_{final} together, denoted as $NoL_{NN_1+NN_{final}}$.

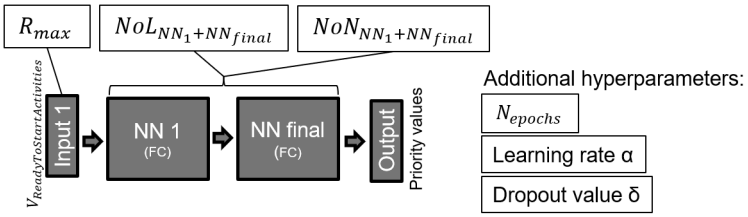

Figure 4.37: Representation of the fully connected neural network without future resource utilization along with its hyperparameters.

- The total number of neurons per layer of NN_1 and NN_{final} together, denoted as $NoN_{NN_1+NN_{final}}$.
- The number of epochs N_{epochs}.
- The learning rate α.
- The dropout rate σ

The last three hyperparameters are not characteristics of the neural network but are parameters that must be set for the training and have an influence on the final performance on unseen data.

4.3.4 Fully connected neural network with future resource utilization

The second proposed neural network structure is called fully connected neural network with future resource utilization, also denoted as NN-FC-FRU. In contrast to the previous structure, the future resource utilization is also included in the input information and processed by the NN_2 (see Figure 4.38). Since the future resource utilization matrix exhibits a 2-dimensional grid structure of $[K\text{x}T]$, it is predestinated to be processed by a two-dimensional convolutional neural network. After this piece of information has been processed, the output of NN_2 is flattened into a vector and merged with the output of NN_1. NN_{final} takes the merged information as an input and creates the output of the entire neural network.

For what concerns the processing of the future resource utilization matrix, the concepts proposed by Pons, Lidy, and Serra (2016) were considered. In this article, the authors investigated how different dimensions of convolutional filters are able to extract different features from an audio recording matrix. This matrix shows similar characteristics as the future resource utilization matrix.

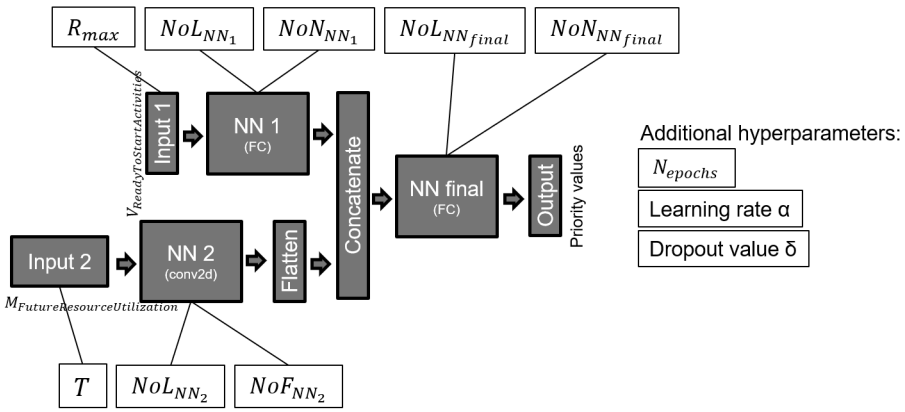

Figure 4.38: Representation of the fully connected neural network with future resource utilization along with its hyperparameters.

While this matrix maps music characteristics defined by the frequency of sounds over a timeline, the future resource utilization matrix maps resource utilizations also on temporal base. Consequently, the concepts developed in this article appear to be transferable for the processing of the information contained in the resource utilization matrix.

Pons, Lidy, and Serra (2016) provides some guidelines for the choice of the filter size and shape to extract the features from a matrix containing temporal information. The idea is to use filters with a width suitable to learn temporal dependencies and with a height suitable to capture the dependencies among the different channels of the temporal information, which in the case of the $M_{FutureResourceUtilization}$ are the K rows representing the K resource types. In this thesis, the convolutional filter shapes to process this matrix were designed based on a combination of the concepts presented in Pons, Lidy, and Serra (2016) and the typical hyperparameter settings for convolutional networks described in Section 3.5.3. Hence, the filter shape in NN_2 has been set to [Kx3].

In Pons, Lidy, and Serra (2016) it is also suggested to additionally use a max-pooling layer with dimension [4x1] after each convolutional layer. However, it has been decided not to follow this suggestion due to the limited length of the input matrix $M_{FutureResourceUtilization}$.

For what concerns the information processing in NN_2, the parameter configuration can be summarized in this list:

- Filter size of [Kx3]. The filter height is set equal to the number of different resource types. As a result, some filters are able to consider all the rows of the matrix at once.
- Strides s=1.
- A zero-padding is used and the padding p is set in such a way that the shrinkage of the processed information throughout the layers is prevented, which is essential to be able to create a deep network.

This parameter configuration for the NN_2 is extended also to the others proposed neural network structures that use the $M_{FutureResourceUtilization}$ in the input information (see Section 4.3.6 and 4.3.8).

The hyperparameters of this neural network structure are shown in Figure 4.38 and are the following ones:

- The maximum number of considered "ready to start" activities R_{max}.
- The time horizon T.
- The number of fully connected layers in NN_1, denoted as NoL_{NN_1}.
- The number of neurons per layer of NN_1, denoted as NoN_{NN_1}.
- The number of convolutional 2-dimensional layers in NN_2, denoted as NoL_{NN_2}.
- The number of filters per layer of NN_2, denoted as NoF_{NN_2}.
- The number of fully connected layers in NN_{final}, denoted as $NoL_{NN_{final}}$.
- The number of neurons per layer of NN_{final}, denoted as $NoN_{NN_{final}}$.
- The number of epochs N_{epochs}.
- The learning rate α.
- The dropout rate σ.

4.3.5 Convolutional 1-dimensional neural network without future resource utilization

The convolutional 1-dimensional neural network without future resource utilization, also denoted as NN-CONV1D, is a neural network structure which considers only $V_{ReadyToStartActivities}$ as input information. This structure is quite

similar to the one introduced in Section 4.3.3 but, in this case, NN_1 is a convolutional 1-dimensional neural network. As a result, a flatten layer is additionally required after it to transform its output into a vector.

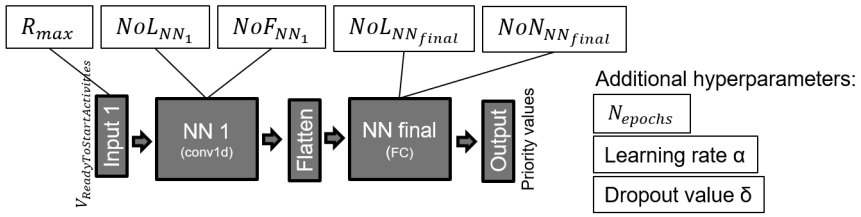

Figure 4.39: Representation of the convolutional 1-dimensional neural network without future resource utilization along with its hyperparameter.

Since NN_1 is not a matrix containing temporal information, different concepts for the choice of the filters are used compared to the ones applied for NN_2. In fact, typical hyperparameter design principles for convolutional networks described in Section 3.5.3 are used. According to those principles, the number of filters should increase and the kernel size should decrease as the information proceeds towards the last layers. For the first layer of NN_1, the author uses a filter size of [(1+K)x1], which is exactly the size of the information about a single "ready to start" activity. For each following layer the filter size is decreased by 1, until a minimum of 3 is reached, and the number of filters is doubled, until the number of filters is equal to four times the initial number. The strides s is set equal to 1, while the padding p is set equal to 0, i.e. no zero are added and the information shrinks layer after layer. However, in NN_1 the information shrinkage is accepted due to the larger input vector size equal to $(1 + K) \cdot R_{max} + K$.

The hyperparameters of this neural network structure are shown in Figure 4.39 and are the following ones:

- The maximum number of considered "ready to start" activities R_{max}.
- The number of fully connected layers in NN_1, denoted as NoL_{NN_1}.
- The number of filters per layer of NN_1, denoted as NoF_{NN_1}.
- The number of fully connected layers in NN_{final}, denoted as $NoL_{NN_{final}}$.
- The number of neurons per layer of NN_{final}, denoted as $NoN_{NN_{final}}$.
- The number of epochs N_{epochs}.
- The learning rate α.

- The dropout rate σ.

4.3.6 Convolutional 1-dimensional neural network with future resource utilization

The convolutional 1-dimensional neural network with future resource utilization, also denoted as NN-CONV1D-FRU, is a neural network structure which considers both $V_{ReadyToStartActivities}$ and $M_{FutureResourceUtilization}$ as input information. This structure is quite similar to the one introduced in Section 4.3.4 but the NN_1 is a convolutional 1-dimensional neural network as for the NN-CONV1D-FRU. A flatten layer is required after both NN_1 and NN_2 to transform the output into a vector.

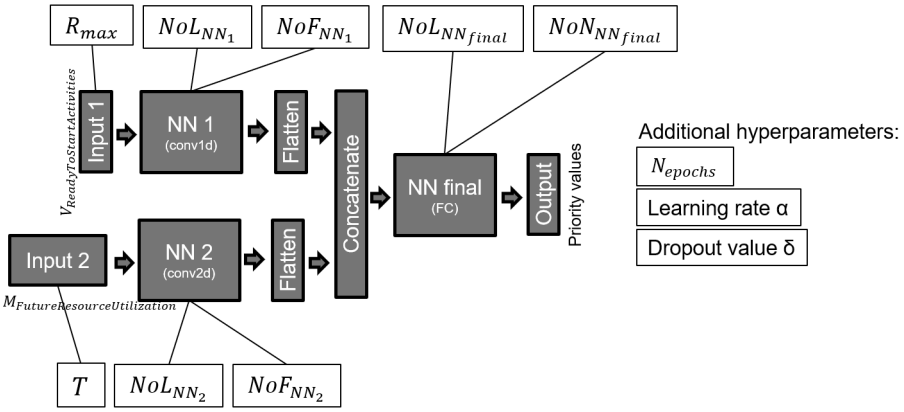

Figure 4.40: Representation of the convolutional 1-dimensional neural network with future resource utilization along with its hyperparameter.

The same considerations done in Section 4.3.5 about the used filters in the NN_1 can be extended also to the NN-CONV1D-FRU neural network.

The hyperparameters of this neural network structure are shown in Figure 4.40 and are listed here:

- The maximum number of considered "ready to start" activities R_{max}.
- The time horizon T.
- The number of fully connected layers in NN_1, denoted as NoL_{NN_1}.
- The number of filters per layer of NN_1, denoted as NoF_{NN_1}.
- The number of convolutional 2-dimensional layers in NN_2, denoted as NoL_{NN_2}.

- The number of filters per layer of NN_2, denoted as NoF_{NN_2}.
- The number of fully connected layers in NN_{final}, denoted as $NoL_{NN_{final}}$.
- The number of neurons per layer of NN_{final}, denoted as $NoN_{NN_{final}}$.
- The number of epochs N_{epochs}.
- The learning rate α.
- The dropout rate σ.

4.3.7 Convolutional 2-dimensional neural network without future resource utilization

The convolutional 2-dimensional neural network without future resource utilization, also denoted as NN-CONV2D, is a neural network structure that considers only $V_{ReadyToStartActivities}$ as input information. This structure is quite similar to the one introduced in Section 4.3.5 but, in this case, NN_1 is a convolutional 2-dimensional neural network and it expects to receive a matrix as an input instead of a vector.

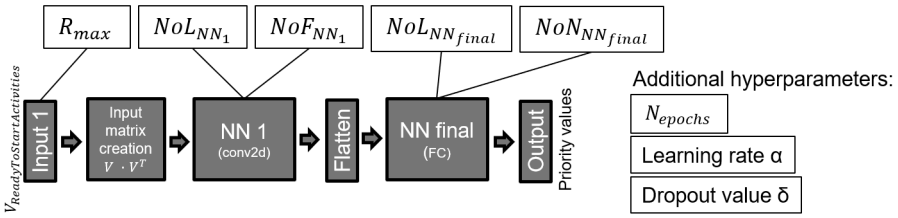

Figure 4.41: Representation of the convolutional 2-dimensional neural network without future resource utilization along with its hyperparameter.

The idea of transforming the input information vector into a matrix and to process it with a 2-dimensional convolutional neural network comes from two major literature contributions, namely Almeida, Lopes, M. A. B. Silva, and Amaral (2018) and Ta and Wei (2018).

In Almeida, Lopes, M. A. B. Silva, and Amaral (2018) a convolutional neural network was utilized for the detection of potential damages in metallic parts. For this purpose, a measurement device produced metrics that were available as a vector which itself served as an input to a convolutional neural network. Almeida, Lopes, M. A. B. Silva, and Amaral (2018) compared two different concepts of processing the data vector with a convolutional neural network. For

their first concept, they kept the vector as it is and fed it into a 1-dimensional convolutional neural network, while in their second concept it is transformed into a squared matrix by dividing the vector into equally sized segments and stacking the segments to a matrix as it is illustrated in Figure 4.42. To overcome the fact that no exact square matrix dimension could be achieved with their vector length, the authors applied zero padding at the end of the vector so that the square root of the vector length L is an integer number. Once the matrix was created, they processed the matrix with a 2-dimensional convolutional neural network.

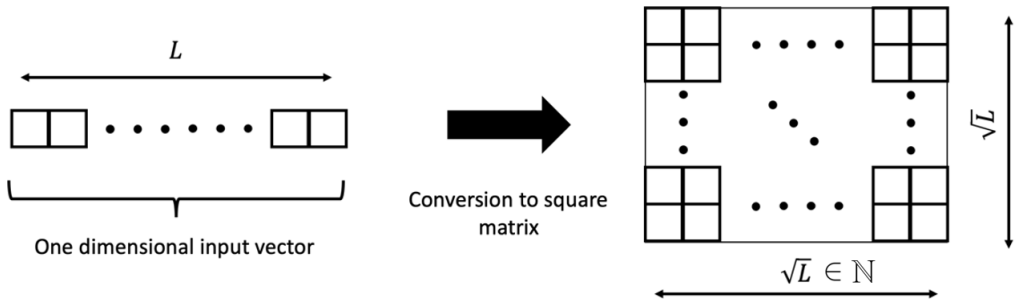

Figure 4.42: Conversion of a vector into a matrix though segmentation and stacking (Almeida, Lopes, M. A. B. Silva, and Amaral (2018)).

On the other hand, Ta and Wei (2018) predicted oxygen levels in the water of aquaculture systems by feeding aqua metrics into their convolutional neural model. Since the measured metrics were originally available as a vector, they converted it into a matrix so that a two-dimensional convolution operation could be employed. The input matrix was generated by multiplying the input vector with its transposed. This input matrix then served as the input to their model. The generation of the input matrix is formally generated with equation 4.14.

$$M_{Input} = V_{Input} \cdot V_{Input}^{T} \tag{4.14}$$

Inspired by the ideas of the two articles, the two additional neural network structures, one in this section and one in Section 4.3.8, are proposed in this thesis where the input vector $V_{ReadyToStartActivities}$ is converted into an input matrix. Since the dimension of the current state vector is variable and depends on R_{max}, zero-padding would have been necessary in most of the times to transform the vector into a matrix when using the approach of Almeida, Lopes, M. A. B. Silva, and Amaral (2018). Thus, it was regarded as more adequate to use the method

presented in Ta and Wei (2018) where the input vector is multiplied by its transpose. This second approach prevent a potential dilution of the information caused by zero-padding.

Hence, the $M_{ReadyToStartActivities}$ is generated by multiplying the input vector $V_{ReadyToStartActivities}$ by its transposed as shown in equation 4.15.

$$M_{ReadyToStartActivities} = V_{ReadyToStartActivities} \cdot V^{T}_{ReadyToStartActivities} \tag{4.15}$$

For what concerns the choice of the filter size the same considerations done in Section 4.3.5, can be extended to a 2-dimensional input and convolutional neural network. Also in this case, the number of filters is increased and the kernel size is decreased as the information proceeds towards the last layers of NN_1. For the first layer, a square filter size of [(1+K)x(1+K)], which is exactly the size of the information about a single "ready to start" activity. For each following layer the filter size is decreased by 1 in both height and width, until a minimum of 3 is reached, and the number of filters is doubled, until the number of filters is equal to four times the initial number. The strides s is set equal to 1, while the padding p is set equal to 0, i.e. no zero are added and the information shrinks layer after layer.

The hyperparameters of this neural network structure are shown in Figure 4.41 and are the listed here:

- The maximum number of considered "ready to start" activities R_{max}.
- The number of fully connected layers in NN_1, denoted as NoL_{NN_1}.
- The number of filters per layer of NN_1, denoted as NoF_{NN_1}.
- The number of fully connected layers in NN_{final}, denoted as $NoL_{NN_{final}}$.
- The number of neurons per layer of NN_{final}, denoted as $NoN_{NN_{final}}$.
- The number of epochs N_{epochs}.
- The learning rate α.
- The dropout rate σ.

4.3.8 Convolutional 2-dimensional neural network with future resource utilization

The convolutional 2-dimensional neural network with future resource utilization, also denoted as NN-CONV2D-FRU is a neural network structure which considers both $V_{ReadyToStartActivities}$ and $M_{FutureResourceUtilization}$ as input information. This structure is quite similar to the one introduced in Section 4.3.6 but, in this case, NN_1 is a convolutional 2-dimensional neural network which requires a matrix as an input. Consequently, the input vector conversion into a matrix introduced in Section 4.3.7 is required.

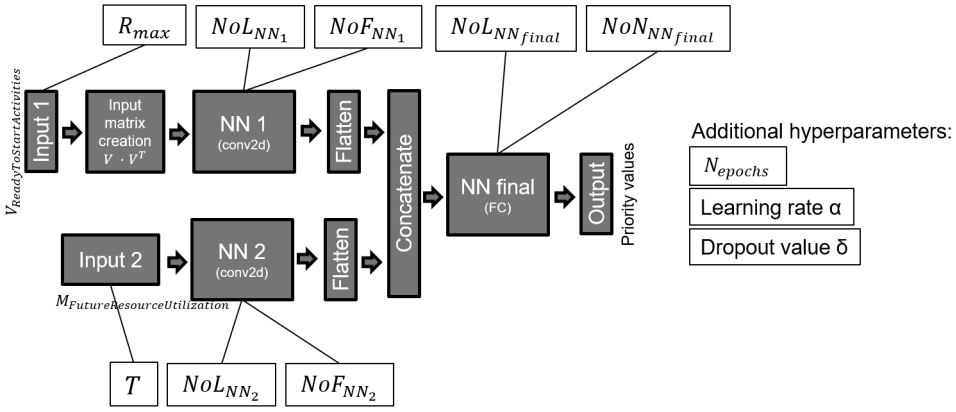

Figure 4.43: Representation of the convolutional 2-dimensional neural network with future resource utilization along with its hyperparameter.

The hyperparameters of this neural network structure are shown in Figure 4.40 and are listed here:

- The maximum number of considered "ready to start" activities R_{max}.
- The time horizon T.
- The number of fully connected layers in NN_1, denoted as NoL_{NN_1}.
- The number of filters per layer of NN_1, denoted as NoF_{NN_1}.
- The number of convolutional 2-dimensional layers in NN_2, denoted as NoL_{NN_2}.
- The number of filters per layer of NN_2, denoted as NoF_{NN_2}.
- The number of fully connected layers in NN_{final}, denoted as $NoL_{NN_{final}}$.
- The number of neurons per layer of NN_{final}, denoted as $NoN_{NN_{final}}$.
- The number of epochs N_{epochs}.

- The learning rate α.
- The dropout rate σ.

4.4 Performance measurement

The goal of this thesis is not only to propose neural network-based scheduling policies but also to quantitative evaluate their performances for a large set of projects and compare them to the ones of other scheduling algorithms. In order to do it, a performance indicator PI_{Π} is required. In Section 2.8, it has been mentioned that the most common PI_{Π} is the one based on the unreachable lower bound for the project's duration (or makespan), which is computed with the lower bound critical path method (LB-CPM) (see Equation 2.5). Its main advantage is that it can be quickly and exactly computed. However, it also has some drawbacks. Since the lower bound is unreachable, it gives a wrong feeling about how far away the proposed scheduling policy is from the unknown optimal solution. Secondly, it does not provide a direct information about how much the neural network has learned so far which is an important indicator in machine learning. Thirdly, the sum of the absolute gaps penalizes the improvements on inherently faster projects although, according to the author, a 10% reduction of the project's duration should be equally rewarded regardless of the expected duration with a RAN policy.

Consequently, the author of this thesis has decided to use a different performance indicator, namely the one presented in equation 4.16. Instead of tracking the distance from an unreachable lower bound, the distance from an upper bound is measured. The upper bound is computed using the random policy RAN and by using 10.000 runs to decrease the confidence interval. The adequacy of this number of runs is tested in section 5.5.2. Moreover, the gaps from the upper bound are computed in percentage which solves the third problem introduced in the previous paragraph.

$$AIP_{\Pi} = PI_{\Pi} = \frac{\sum_{p \in P} \frac{MS_{p,RAN} - MS_{p,\Pi}}{MS_{p,RAN}}}{P} \tag{4.16}$$

The new performance indicator is also denoted as the average improvement percentage compared to the random policy and it called AIP in this thesis. By using this indicator, the learning progress of a neural network can be visualized

very well. If the training process is performed correctly and the performance are measured on unseen projects, it is expected that the AIP_{Π} on these projects starts in a random point of an interval centered in zero immediately after the random weight initialization, progressively increases, reaches the highest point after some epochs and decreases afterwards when overfitting occurs.

4.5 Neural network design methodology

As mentioned in section 2.7, the scope of this thesis is to design a set of deep neural networks able to schedule the activities in an RCPSP problem. The chosen machine learning approach is supervised learning but, since there is no preexisting labeled dataset defining which set of activities should be started at different decision points, it must be created. The second major problem is that the performance of the decision policy on unseen projects cannot be measured as the percentage of decision situations in which the neural network produces the optimal output since the optimal output is unknown for projects with more than 30-50 activities (Abdolshah (2014)) and more than one optimal output could exist. Thirdly, a lot of hyperparameters are involved in the neural network design and a rigorous procedure is required to choose them. The following sections present the methodology used to tackle all those problems and to design a neural network-based decision tool.

4.5.1 Training data generation

In Section 3.7 the problem of object grasping has been shortly mentioned. In this kind of problem, for instance, the input information is an image, while the output of the decision model is a vector whose elements define how the object should be grasped. In order to train a neural network model which is able to autonomously grasp objects, each available image must be labeled with the correspondent target values for the grasping parameters. In case of a reactive policy able to autonomously identify the starting activities at each decision point, the input information is given by the current state of the activity sequence at the beginning of each step 1 (see Section 4.1.1). This can be described by the vector $V_{ReadyToStartActivities}$ and by the matrix $M_{FutureResourceUtilization}$. Assuming that the optimal set of "ready to start" activities to begin with is known, each

decision state can be labeled with a vector of zeros and ones as explained in Section 4.3.2 creating the so-called state-action pairs.

Figure 4.44 shows how the state-action pair for the decision situation introduced in Figure 4.27 looks like, if the neural network should learn to start activity 3.

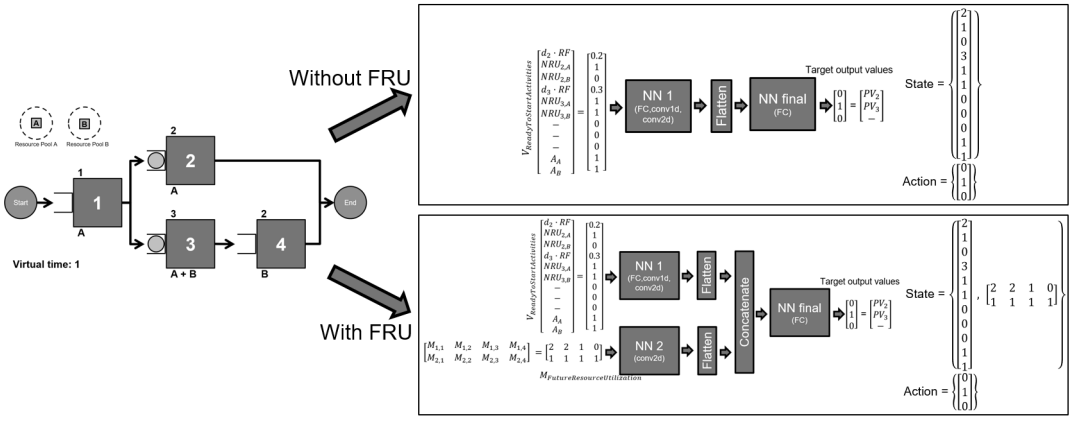

Figure 4.44: State actions pairs for an example of decision point considering a neural network structure with and without future resource utilization.

The problem of not having training data implies that there are neither preexisting decision state matrices nor the correspondent target action vectors. Both must be generated, for instance, by means of simulation, which is the chosen method in this thesis. In order to generate the training data considering a wide spectrum of decision situations and, consequently, to train a neural network able to generalize well, a large set of randomly generated activity sequence instances are required. Those sequences are called training projects. For each of them, N independent simulation runs with the random policy RAN are performed and the run with the lowest makespan is considered (N=10.000 is the default value for the results of this thesis). This simulation run is characterized by a series of decisions taken in different intermediate states. Those states along with the correspondent decisions are identified, converted in state-action pairs and added to the training dataset (see Section 4.5.2).

During the simulation run, it is possible that some trivial decision situations are encountered. Considering a greedy reactive scheduling policy (see Section

4.2.2), a trivial decision situation is defined as a situation where one of the following conditions is true:

- There are no "ready to start" activities, i.e. the current resource availability and/or the precedence constraints do not allow any "idle" activity to start.
- There are enough resources to start all "ready to start" activities.

In both situation, the assigned priority values are irrelevant. In the first case, the priority values are assigned to dummy placeholders in neural network output but no real activities are eligible to start, while, in the second case, the activity order is irrelevant since all of them will be started at the end of the decision process. As a result, the state-action pairs of trivial decision situations are not added to the training data.

Instead of following a traditional supervised learning approach where the state-action pairs dataset would be split into training, validation and test set and the performance would be measured as the percentage of unseen decision situations where the neural network produces the optimal output, the entire database is used for the training. The reasons why a traditional approach is not suitable are, firstly, that one cannot be sure that all labels are correct and, secondly, that the average improvement percentage AIP (see Section 4.4) expresses much better the goodness of the decision tool in minimizing the makespan of unseen projects which is the ultimate goal. Consequently, the validation and test phases are not done on state-action pairs but on new randomly generated activity sequences, called respectively validation and test projects.

4.5.2 Tuning of the hyperparameter

Once the training dataset has been created, it is possible to move on to the second step, namely the hyperparameter tuning. This step is not compulsory since it is also possible to guess reasonable hyperparameters considering the task complexity and to move on to the final step, i.e. the performance measurement of the chosen neural network on the test projects after training. However, the first guess on the hyperparameters may be quite different from the optimal one.

The hyperparameter tuning let the user test different combinations and choose the best one according to an objective function (see Section 3.2.5) which is the average improvement percentage AIP on the validation projects in the case.

For this thesis, the hyperparameter tuning is automatic (no manual choice of the parameter combinations to sample) and based on a Bayesian optimization. The latter enables to choose the next combinations to sample without a human intervention.

This kind of hyperparameter tuning can be schematized as shown in 4.45. Two different groups of hyperparameters can be distinguished, namely the ones to be tuned, denoted as the set $\mathcal{H}$, and the fixed ones, denoted as the set $\bar{\mathcal{H}}$.

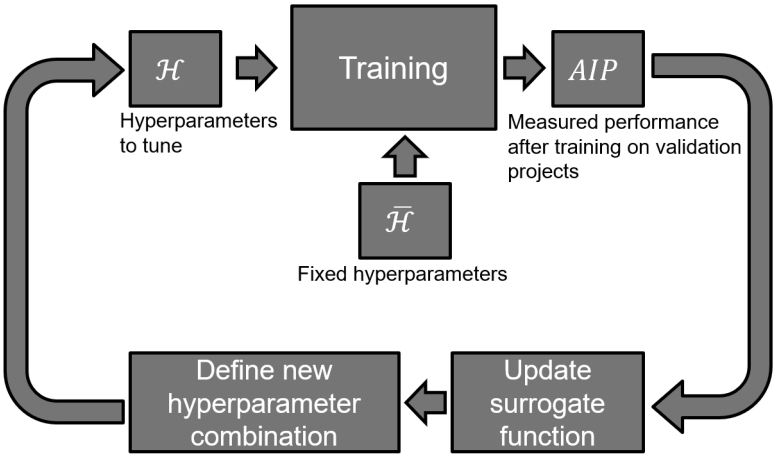

Figure 4.45: Scheme of the hyperparameter tuning.

Depending on how many possible hyperparameters the neural network could have and how time-consuming an evaluation can be, the user can choose to include a different number of hyperparameters in the set $\bar{\mathcal{H}}$. In this thesis, the ones to be tuned have been introduced in the Sections from 4.3.3 to 4.3.8, while the following ones are kept fixed:

- **Optimizer**. As mentioned in 3.6, the neural network is trained by using iterative gradient descent-based optimization algorithms that drive the loss function to a low value by repeatedly adjusting the weights w. The Adam optimization algorithm has been chosen in this thesis since it is extremely popular, it incorporates most of the advantages of other gradient descending-based optimization algorithms and often performs very competitively Aggarwal (2018).
- **Weights initialization**. Before the training process is started, some random initialization values must be assigned to the weights of the neural network. In this thesis, it has been assumed that the probability distribution from which the values are randomly generated does not play a major

role on the performance after the training if the Adam optimization works properly. As a result, a truncated normal distribution has been chosen for all training.

- **Activation function**. As mentioned in Section 3.3.3, different activation functions can be used in the neurons and/or in the filters of the neural network. The ReLU has been chosen since it is currently the most widely used activation function in convolutional layers (Krizhevsky, Sutskever, and Hinton (2012)).
- **Max pooling for convolutional layers**. As mentioned in Section 3.5.1, a typical convolutional layer includes three stages. The last one is called pooling stage and it is where the pooling function modifies the output of the previous stage by grid-wise summarizing it. This stage is not used for the neural networks proposed in this thesis due to the small size of the input matrices and due to the information loss which is generally associated to this operation.

Actually many hyperparameters could be defined, like for example a parameter defining how the number of neurons of filters changes layer by layer within the same neural network. However, they are not mentioned in this thesis since their enumeration is beyond the scope of this thesis.

As shown in Figure 4.45, the testing of different hyperparameter combination is done in series, which means that, as soon as an evaluation is completed, the surrogate function is updated and the next combination is chosen for the next evaluation. The used stopping criteria of the hyperparameter tuning is the number of evaluated combinations which has been set to 50.

The trained neural network with the best performance indicator AIP on the evaluation projects is saved and used for the final evaluation of the correspondent neural network structure on the test projects in the next step.

4.5.3 Evaluation on the test projects

Once the best hyperparameter configuration among the 50 evaluated ones has been identified, the correspondent trained neural network model is deployed for the scheduling of the test projects. The validation projects cannot be used for the final evaluation since the chosen hyperparameters generally overfit their

decision situations. As a result, a lower value for the performance indicator is expected when the trained neural network is tested on test projects.

This last statement along with all considerations of Section 4.5 are valid under the two following conditions:

- The projects of the three sets (training, validation and test set) have been randomly generated with the same rules, i.e. they belong to the same project class.
- The number of projects in each set is large enough.

For what concerns the first condition, it is important to underline that the neural network can learn to properly schedule activities only by situations that occur in the class of RCPSP problem defined by the training projects. If the validation and test projects or all future projects that the neural network will be asked to schedule belong to another class, poor results may be achieved.

Assuming that the projects of the three sets belong to the same class, there is still the risk that too few training, validation and test projects are considered. In all three cases, the project set must be representative of the entire spectrum of possible decision situations related to this project class.

4.6 Considered class of projects and project generator

At this point, it is clear that the proposed methodology requires a large number of activity sequences. Most of the literature contributions use predefined project sets that have been generated with the two most popular random project generators, i.e. the ProGen and the RanGen2 (see Section 2.8). The first main limitation of these project sets, i.e. J30, J60, J120 and RG30, is a limited number of instances, which is between 480 and 1800. As shown in Section 5.5.4, a much bigger number of project instances is required to train a neural network that generalizes well and achieve a good performance on unseen projects. Moreover, the number of activities per sequence is not varied within the same set and it may be possible that the neural network only learns to schedule well with projects of a particular class.

The second possible approach is to use the ProGen or the RanGen2 generation tool to create any number of projects according to user-defined rules. One of

the major differences between the two project generation tools is that, while the RanGen2 can be considered as strongly random generator since each project topology is the result of a random selection from the space of all possible networks which satisfy the user-defined input parameters (Demeulemeester, Vanhoucke, and Herroelen (2003)), the ProGen considers some following additional constraints, which are not user-defined:

- Each project starts with exactly three activities (after the dummy start activity).
- Each project ends with exactly three activities (before the dummy end activity).
- Each activity is followed by a number of activities between one and three. The last three activities in the sequence are excluded from this constraint.
- Each activity is preceded by a number of activities between one and three. The first three activities in the sequence are excluded from this constraint.

Without these additional constraints, the user must pay more attention during the setting of the parameters. For example, it may happen that the RanGen2 generates sequences with more than half of the activities placed just after the start dummy activity which is quite unrealistic. As a result, the format of the ProGen seems preferable to the author of this thesis.

While RanGen2 is available on the *Random Network Generation Website* (2020) in an up-to-date version for Windows, only an old version of the ProGen in Turbo Pascal is available by request. This version was used in the late 90's (Kolisch and Hartmann (1999)) to generate the J30, J60 and J120 project sets.

Therefore, the author of this thesis has decided to develop his own project generator (available at *Repository of the P-30-120-10 project generator and test projects* (2021)) with generation rules that resembles the ones of the ProGen. One of the major advantages of developing his own project generator is the possibility to integrate also new customizable features. For example, in most of the literature contributions on the RCPSP, the performance of a scheduling policy is evaluated on groups of projects with the same number of activities and the available project generators were not designed to randomize also number of activities in the project. However, this feature is quite useful for the scope of this thesis since the goal of this thesis is to design a neural network able to schedule the activities in projects of different sizes.

In this thesis, 50.000 project instances of a new project class with the following characteristics have been created:

- The number of activities J is sampled from a discrete uniform distribution with parameters J_{min}=30 and J_{max}=120.
- Each project starts with exactly three activities (as for the ProGen).
- Each project ends with exactly three activities (as for the ProGen).
- Each activity is followed by a number of activities between one and three. The last three activities in the sequence are excluded from this rule (as for the ProGen).
- Each activity is preceded by a number of activities between one and three. The first three activities in the sequence are excluded from this rule (as for the ProGen).
- The duration of each activity d_j is sampled from a discrete uniform distribution with parameters d_{min}=1 and d_{max}=10 (as for the ProGen).
- K=4 types of resources are considered (as for the ProGen).
- The total number of resource units R_k for each resource type is set to R_{all}=10 for each k (as for the ProGen).
- The maximal number of required resource types K_{max} per activity is sampled from a discrete uniform distribution with parameters $K_{max,min}$=1 and $K_{max,max}$=K. This parameter determines how many different resource types the project activities can require at maximum by each activity and it is a property of the project.
- Each activity only requires K_j different resource types. The value of K_j is sampled from a discrete uniform distribution with parameters 1 and K_{max}. If K_j=1, each activity requires only one resource type. If K_j=2, each activity requires two resource types and so on. The K_j required resource types are randomly chosen among the K resource types for each activity.
- The resource consumption $r_{j,k}$ of activity j and required resource type k is sampled from a discrete uniform distribution with parameters r_{min}=1 and r_{max}=R_{all}.

Due to its characteristics the project class has been denoted as P-30-120-10 according to the following notation: $P - J_{min} - J_{max} - R_{all}$.

After the project generation, the projects have been randomly divided with a 80:10:10 split, namely 40.000 training projects, 5.000 validation projects and 5.000 test projects. The test projects used in this thesis are available at *Repository of the P-30-120-10 project generator and test projects* (2021) as well.

5 Evaluation

Leaving society to algorithms will
be like leaving healthcare to
stethoscopes.
-A. Naskar

In this chapter, the numerical results are presented. First of all, the results of the hyperparameter tuning for all six neural network structures are analyzed and, secondly, the performance comparison with other reactive policies is presented. In the final part of the chapter, the neural network with the best performance after the hyperparameter tuning and the training is compared more in detail with the best considered reactive policy and the trade-off between the performance indicator AIP, which is related to the scheduling performance, and the decision time is analyzed.

In the performance comparisons of this chapter, the confidence intervals for the AIP values are intentionally omitted. However, due to the deterministic nature of the considered problem, the only source of variability is the computation of the project's total durations with the baseline scheduling policy RAN (random priority values for the "ready to start" activities) based on which the AIP values for the other scheduling policies are computed. Thanks to the large number of project instances and the large number of simulation runs used to estimate the performance of the RAN policy (see section 4.2.3), the variance is very small compared to the scale of the diagrams and the confidence intervals would be hardly visible. As a result, they have been omitted. However, considering 95% confidence intervals created 5 samples, it is possible to state that all confidence intervals for the AIP indicator of the scheduling policies in this chapter are less or equal than 0.0129%.

5.1 Used hardware and software

As some of the results of this chapter, especially the ones regarding the computing times, are influenced by the used hardware, software and libraries, they are explicitly listed in the following paragraphs.

5.1.1 Hardware

Starting from the hardware, a desktop computer with the following characteristics has been used:

- CPU: AMD Ryzen 7 2700X, 8 cores (16 threads), 3.70GHz (max. 4.30GHz).
- RAM: 4x16GB G.Skill Aegis DDR4-3000 DIMM CL16.
- GPU: MSI Nvidia GeForce RTX 2080, 3072 CUDA cores, 1845 MHz memory speed, 8 GB of RAM.

5.1.2 Programming language

The simulation environment is programmed in the programming language Python (version 3.6.8). Python is widely used both in business as well as in research fields (Lutz (2013)). The syntax of Python is pseudocode-like which makes it easy to read and reduces the extent of the code by one third compared to other programming languages like C, C++ or Java (Lutz (2013)). While low level programming languages like C or C++ obtain better performance out of the hardware they are running on, Python usually has the more flexible and extensible code (Johansson (2019)). Python is especially widely used in the field of machine learning because of the availability of many free and open-source machine learning libraries like, for example, TensorFlow.

5.1.3 Used libraries

Released by Google in 2015, TensorFlow is an open-source machine learning software library for defining, training and deploying machine learning models. In TensorFlow, machine learning algorithms are expressed as computational graphs where vertices describe operations and edges represent data flowing between these operations. The name TensorFlow is derived from the tensor-shaped

data flowing over the edges from one operation to another operation (Goldsborough (2016)). Although many other alternatives are also freely available, like for example Theano, Torch and Caffe, Tensorflow (GPU version) have been chosen to generate the results of this thesis. Its ability to perform fast automatic gradient computation, its inherent support for distributed computation and its powerful visualization tools make it a very suitable to the field of machine learning. On one hand, it provides low-level programming interfaces which imply a very good control on the neural network construction, while abstraction libraries built on top of it (e.g. TFlearn) allow the user to quickly build and modify models in few lines of code. The highly modular interface facilitates the rapid chaining of neural network layers and the changing of the model's hyperparameters without losing most of functionalities provided by Tensorflow. The neural networks that have been constructed in the course of this thesis have all been designed with TFLearn v0.5.0 (based on tensorflow v1.12). Compared to other machine learning frameworks, it is a good trade-off between user-friendliness and computing speed (Goldsborough (2016)).

Further used libraries which are noteworthy are:

- Numpy v1.16.4 (*Library numpy* (2020)). It supports the operations with multi-dimensional matrices and is mandatory for the use of Tensorflow.
- Multiprocessing v2.6.2 (*Library multiprocessing* (2020)). It enables the possibility to run the same function in parallel on more than one core with different input arguments. This library is particularly useful during the training data generation since the simulations can be run in parallel.
- Hyperopt (*Library hyperopt* (2020)). It is a package for the implementation of Bayesian optimization algorithms based on Gaussian processes and regression trees for Python (Bergstra, Yamins, and Cox (2013)).

All pieces of software and libraries have been installed on a machine running an Ubuntu 18.04.5 LTS operating system

5.2 Default settings for the evaluations

This section presents the default settings for all experiments and evaluations of the chapter. In fact, this default configuration is used most of the times and only few parameters are varied. In order not to repeat the value of each

parameter used in each experiment or evaluation, the following default values for the parameters are outlined in the following paragraphs.

For what concerns the project database, 50.000 projects of the class P-30-120-10 are considered, i.e. with a number of activities between 30 and 120 and a total resource availability of 10 for each resource type. They have been generated with the generator presented in Section 2.8. The projects are divided in the training, validation and test set with an 80:10:10 split.

For each considered reactive scheduling policy, a greedy decision process as explained in Section 4.2.2 is considered. That implies that if an activity is considered in the decision process, all previous activities have been completed and there are enough resources to start it, it is always decided to start it.

The parameter N of the lower bound scheduling policy LB-N is set to 10.000, which means that 10.000 simulation runs with a random policy are performed, the best one is considered and the activities scheduled at the decision time t_d are started.

For what concerns the neural network-based decision policies, the training data are also created with 10.000 simulation runs with a random policy as for LB-N. The training data and the input information of the neural network are always normalized and both the resource and activity conversion is always used. Lastly, no max-pooling is used within the layers.

5.3 Hyperparameter tuning for the chosen neural network configurations

During the generation of the training data with the methodology described in Section 4.5.1, 1.077.246 state-action pairs are obtained. With these training data it is possible to move on to the second (optional) step which is the hyperparameter tuning (see Section 4.5.2) for all six considered neural network structures.

5.3.1 Hyperparameter tuning for the fully connected neural network without FRU

The first neural network structure to tune is the fully connected neural network without future resource utilization, also denoted as NN-FC. The considered

hyperparameters in the tuning process have been already introduced in Section 4.3.3 and they are depicted in Figure 5.1.

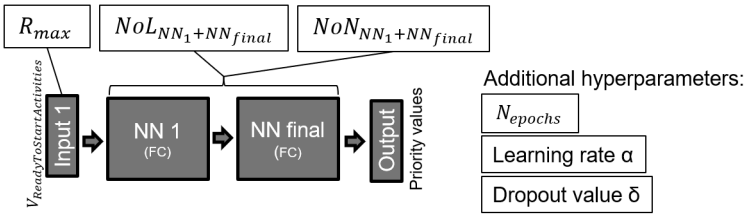

Figure 5.1: Representation of the fully connected neural network without future resource utilization along with its hyperparameters.

In order to run a hyperparameter tuning, the minimum and maximum value for each parameter along with the possible intermediate ones must be defined. By doing this, the search space is defined and delimited. Table 5.1 shows which parameter values belong to the search space and the best combination that the Bayesian optimization has found with 50 iterations. In particular, for what concerns the maximum number of considered "ready to start" activities R_{max}, the number of layers $NoL_{NN_1+NN_{final}}$, the number of epochs N_{epochs} and the dropout value σ, equally spaced values have been considered, while powers of 2 and powers of ten have been considered for the number of neurons per layer $NoN_{NN_1+NN_{final}}$ and for the learning rate α respectively.

Table 5.1: Hyperparameter ranges for the tuning of a fully connected neural network without future resource utilization

Hyperparameter	Min value	Value of the best configuration	Max value	Considered values
R_{max}	6	7	9	[6,7,8,9]
$NoL_{NN_1+NN_{final}}$	8	9	12	[8,9,...,11,12]
$NoN_{NN_1+NN_{final}}$	256	512	1024	[256,512,1024]
N_{epochs}	10	14	25	[10,11,...,24,25]
α	10^{-5}	$10^{-4.5}$	10^{-3}	[10^{-5},$10^{-4.9}$,...,$10^{-3.1}$,10^{-3}]
σ	0.5	0.71	0.9	[0.5,0.51,...,0.89,0.9]

Considering the best combination, it can be noticed that some of its values coincide with one of the extreme values of the intervals. In this case, it could be advisable to change the interval, center it on the chosen value and rerun the tuning to understand if even better values lay outside the initially chosen

interval. However, in order to maintain comparable parameter ranges among all neural network types, it has been decided not to adjust them.

Figure 5.2 represents the learning curve, i.e. the performance in terms of AIP indicator on the validation projects after each epoch of training, for the best considered hyperparameter configuration. It is important to notice that the performance never systematically decreases which means that no overfitting occurred.

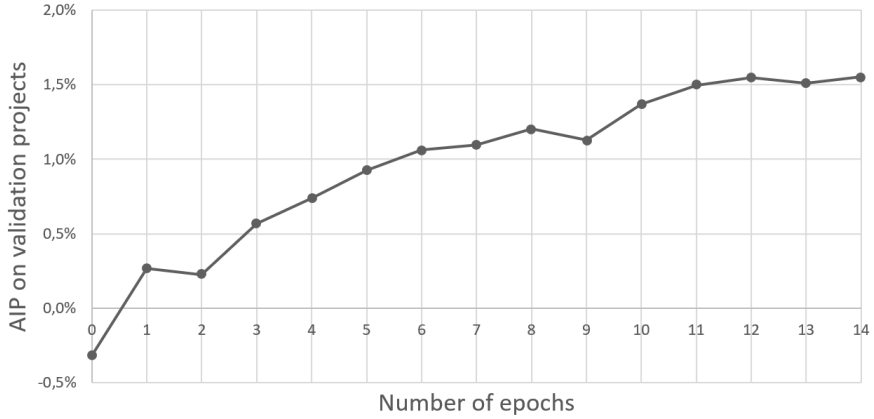

Figure 5.2: Learning curve of the fully connected neural network without FRU.

The hyperparameter tuning with 50 sampled combinations for this neural network structure took 11.46 days.

5.3.2 Hyperparameter tuning for the fully connected neural network with FRU

The second neural network structure to tune is the fully connected neural network with future resource utilization, also denoted as NN-FC-FRU. The considered hyperparameters in the tuning process have been already introduced in Section 4.3.4 and they are depicted in Figure 5.3.

As for the previous neural network structure, the search space must be defined and delimited before starting with the hyperparameter tuning. Table 5.2 shows which parameter values belong to the search space and the best hyperparameter combination that the Bayesian optimization has found. In comparison to the previous neural network structure, there are a couple of additional hyperparameters, namely the time horizon T and the number of layers and filters in the neural network processing the future resource utilization matrix NoL_{NN_2} and

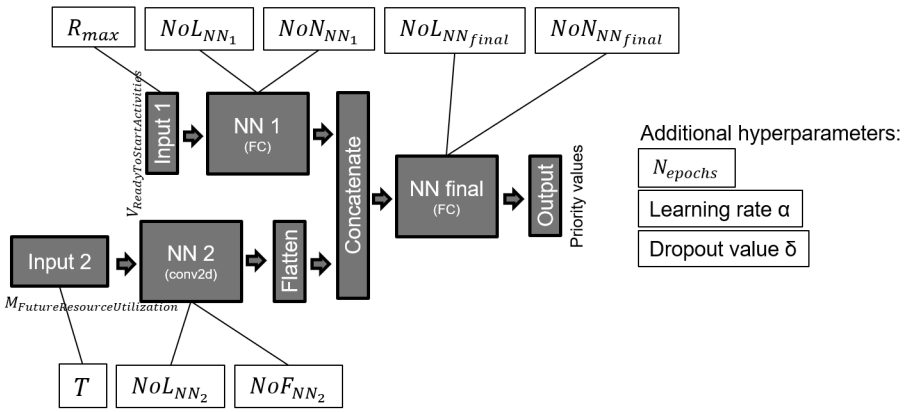

Figure 5.3: Representation of the fully connected neural network with future resource utilization along with its hyperparameters.

NoF_{NN_2}. For the first two parameters, equally spaced values have been chosen, while powers of 2 have been chosen for the number of filter. For what concerns the other parameters, the same considerations done in Section 5.3.1 still hold.

Table 5.2: Hyperparameter ranges for the tuning of a fully connected neural network with future resource utilization

Hyperparameter	Min value	Value of the best configuration	Max value	Considered values
R_{max}	6	6	9	[6,7,8,9]
T	5	12	15	[5,6,...,14,15]
NoL_{NN_1}	4	4	6	[4,5,6]
NoN_{NN_1}	256	1024	1024	[256,512,1024]
NoL_{NN_2}	4	6	6	[4,5,6]
NoF_{NN_2}	32	64	128	[32,64,128]
$NoL_{NN_{final}}$	4	6	6	[4,5,6]
$NoN_{NN_{final}}$	256	256	1024	[256,512,1024]
N_{epochs}	10	15	25	[10,11,...,24,25]
α	10^{-5}	10^{-4}	10^{-3}	[10^{-5},$10^{-4.9}$,...,$10^{-3.1}$,10^{-3}]
σ	0.5	0.7	0.9	[0.5,0.51,...,0.89,0.9]

Figure 5.4 represents the correspondent learning curve, i.e. the performance in terms of AIP indicator on the validation projects after each epoch of training, for the best considered hyperparameter configuration. Also for this neural network structure, the performance never decreases systematically, which means that probably no overfitting occurred. The performance seems to keep increasing at

the moment when the number of epochs of this evaluation has been reached and, therefore, the training has been stopped. Although better results could be probably achieved with additional epochs, the number of epochs is defined by the hyperparameter optimization at each of the 50 iterations and cannot be changed.

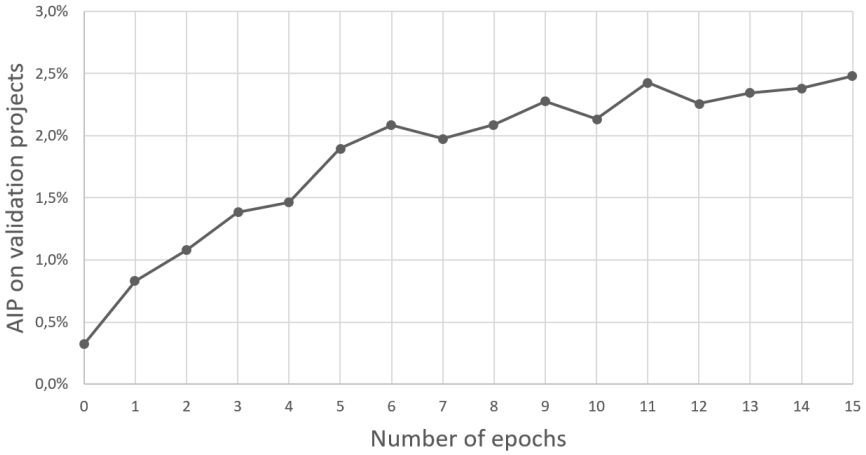

Figure 5.4: Learning curve of the fully connected neural network with FRU.

The hyperparameter tuning with 50 sampled combinations for this neural network structure took 13.67 days.

5.3.3 Hyperparameter tuning for the convolutional 1-dimensional neural network without FRU

The third neural network structure to tune is the convolutional 1-dimensional neural network without future resource utilization, also denoted as NN-CONV1D. The considered hyperparameters in the tuning process have been already introduced in Section 4.3.5 and they are depicted in Figure 5.5.

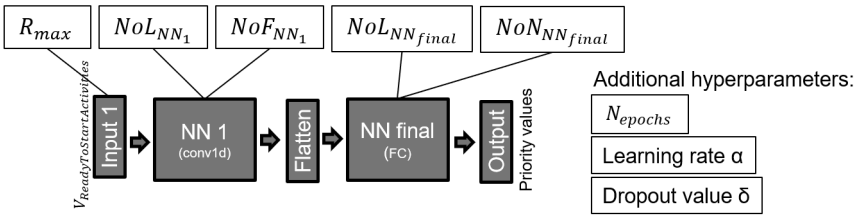

Figure 5.5: Representation of the convolutional 1-dimensional neural network without future resource utilization along with its hyperparameter.

Table 5.3 shows which parameter values belong to the search space and the best combination found by the Bayesian optimization. In comparison to the previous neural network structures, the neural network $NN1$ is processed by convolutional layers instead of fully connected ones and the hyperparameter NoF_{NN_1} is used instead of NoN_{NN_1}. The values for NoF_{NN_1} in the search space have been set to powers of 2 as well.

Table 5.3: Hyperparameter ranges for the tuning of a convolutional 1-dimensional neural network without future resource utilization

Hyperparameter	Min value	Value of the best configuration	Max value	Considered values
R_{max}	6	7	9	[6,7,8,9]
NoL_{NN_1}	4	6	6	[4,5,6]
NoF_{NN_1}	32	128	128	[32,64,128]
$NoL_{NN_{final}}$	4	4	6	[4,5,6]
$NoN_{NN_{final}}$	256	512	1024	[256,512,1024]
N_{epochs}	10	11	25	[10,11,...,24,25]
α	10^{-5}	$10^{-3.9}$	10^{-3}	[10^{-5},$10^{-4.9}$,...,$10^{-3.1}$,10^{-3}]
σ	0.5	0.67	0.9	[0.5,0.51,...,0.89,0.9]

Figure 5.6 represents the correspondent learning curve, i.e. the performance in terms of AIP indicator on the validation projects after each epoch of training, for the best considered hyperparameter configuration. For this neural network structure, the performance seems to reach a plateau and does not decrease after it. As a result, it seems that no overfitting occurred.

The hyperparameter tuning with 50 sampled combinations for this neural network structure took 5.19 days.

5.3.4 Hyperparameter tuning for the convolutional 1-dimensional neural network with FRU

The fourth neural network structure to tune is the convolutional 1-dimensional neural network with future resource utilization, also denoted as NN-CONV1D-FRU. The considered hyperparameters in the tuning process have been already introduced in Section 4.3.6 and they are depicted in Figure 5.7.

Table 5.4 shows which parameter values define the search space and the best hyperparameter combination that the Bayesian optimization has found.

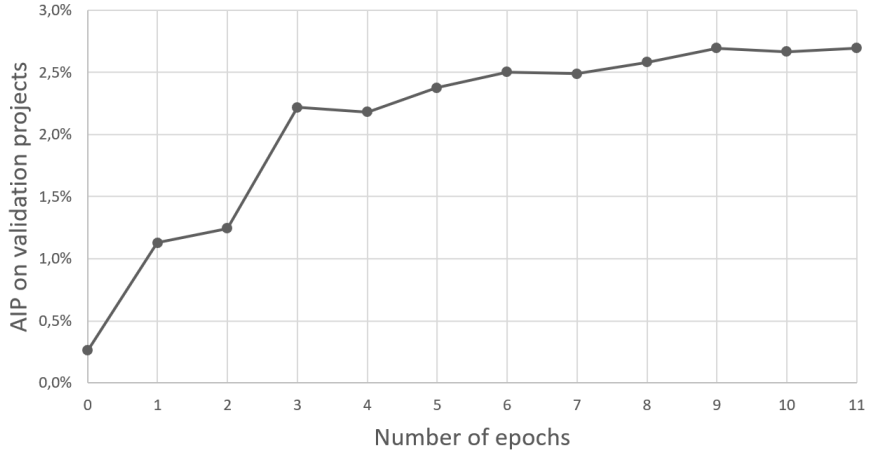

Figure 5.6: Learning curve of the convolutional 1-dimensional neural network without FRU.

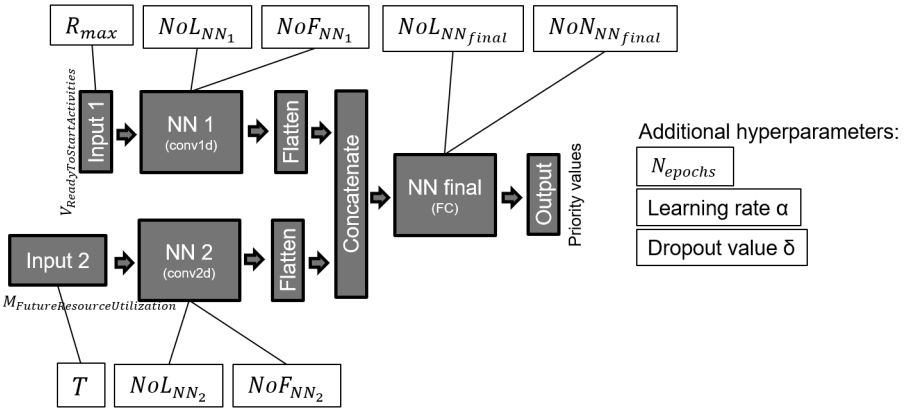

Figure 5.7: Representation of the convolutional 1-dimensional neural network with future resource utilization along with its hyperparameter.

Figure 5.8 represents the correspondent learning curve, i.e. the performance in terms of AIP indicator on the validation projects after each epoch of training, for the best considered hyperparameter configuration. The performance seems to have reached a plateau and does not decrease after it.

The hyperparameter tuning with 50 sampled combinations for this neural network structure took 8.74 days.

Table 5.4: Hyperparameter ranges for the tuning of a convolutional 1-dimensional neural network with future resource utilization

Hyperparameter	Min value	Value of the best configuration	Max value	Considered values
R_{max}	6	6	9	[6,7,8,9]
T	5	14	15	[5,6,...,14,15]
NoL_{NN_1}	4	5	6	[4,5,6]
NoF_{NN_1}	32	128	128	[32,64,128]
NoL_{NN_2}	4	6	6	[4,5,6]
NoF_{NN_2}	32	128	128	[32,64,128]
$NoL_{NN_{final}}$	4	4	6	[4,5,6]
$NoN_{NN_{final}}$	256	512	1024	[256,512,1024]
N_{epochs}	10	11	25	[10,11,...,24,25]
α	10^{-5}	$10^{-4.1}$	10^{-3}	[10^{-5},$10^{-4.9}$,...,$10^{-3.1}$,10^{-3}]
σ	0.5	0.69	0.9	[0.5,0.51,...,0.89,0.9]

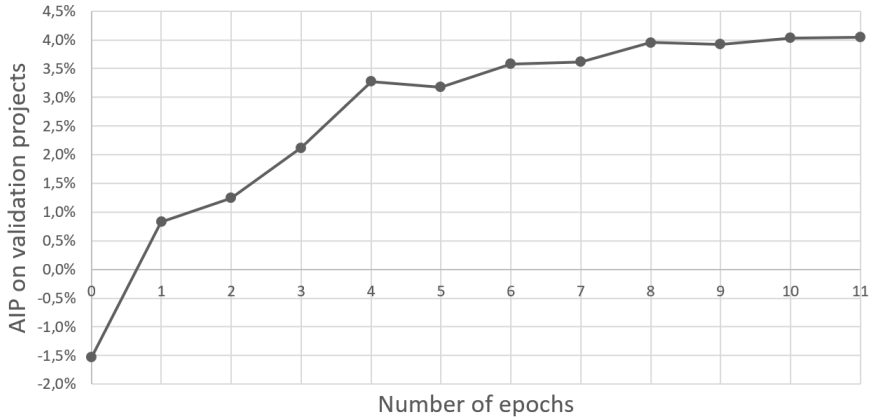

Figure 5.8: Performance of the convolutional 1-dimensional neural network with FRU.

5.3.5 Hyperparameter tuning for the convolutional 2-dimensional neural network without FRU

The fifth neural network structure to tune is the convolutional 2-dimensional neural network without future resource utilization, also denoted as NN-CONV2D. The considered hyperparameters in the tuning process have been already introduced in Section 4.3.7 and they are depicted in Figure 5.9.

Table 5.5 shows which parameter values belong to the search space and the best hyperparameter combination that the Bayesian optimization has found.

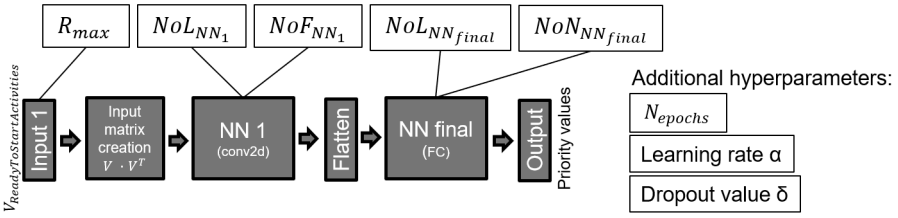

Figure 5.9: Representation of the convolutional 2-dimensional neural network without future resource utilization along with its hyperparameter.

Table 5.5: Hyperparameter ranges for the tuning of a convolutional 2-dimensional neural network without future resource utilization

Hyperparameter	Min value	Value of the best configuration	Max value	Considered values
R_{max}	6	6	9	[6,7,8,9]
NoL_{NN_1}	4	4	6	[4,5,6]
NoF_{NN_1}	32	128	128	[32,64,128]
$NoL_{NN_{final}}$	4	5	6	[4,5,6]
$NoN_{NN_{final}}$	256	512	1024	[256,512,1024]
N_{epochs}	10	10	25	[10,11,...,24,25]
α	10^{-5}	$10^{-3.8}$	10^{-3}	[10^{-5},$10^{-4.9}$,...,$10^{-3.1}$,10^{-3}]
σ	0.5	0.74	0.9	[0.5,0.51,...,0.89,0.9]

Figure 5.10 represents the correspondent learning curve, i.e. the performance in terms of AIP indicator on the validation projects after each epoch of training, for the best considered hyperparameter configuration. Also in this case, the performance seems to have reached a plateau and does not decrease after it, which means that probably no overfitting occurred.

The hyperparameter tuning with 50 sampled combinations for this neural network structure took 12.22 days.

5.3.6 Hyperparameter tuning for the convolutional 2-dimensional neural network with FRU

The fourth neural network structure to tune is the convolutional 2-dimensional neural network with future resource utilization, also denoted as NN-CONV2D-FRU. The considered hyperparameters in the tuning process have been already introduced in Section 4.3.8 and they are depicted in Figure 5.11.

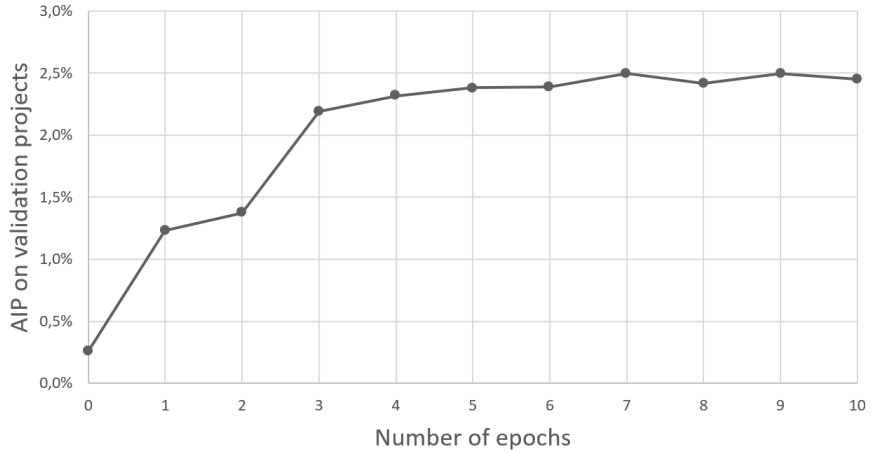

Figure 5.10: Learning curve of the convolutional 2-dimensional neural network without FRU.

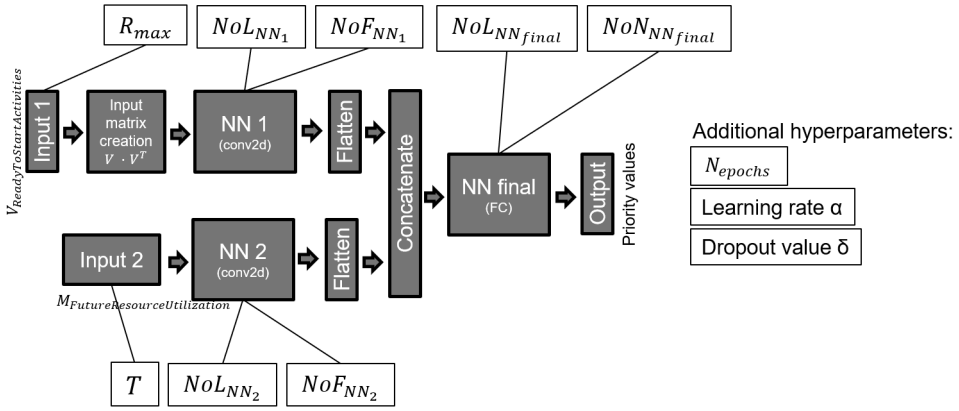

Figure 5.11: Representation of the convolutional 2-dimensional neural network with future resource utilization along with its hyperparameter.

Table 5.6 shows which parameter values define the search space and the best hyperparameter combination found by the Bayesian optimization.

Figure 5.12 represents the correspondent learning curve, i.e. the performance in terms of AIP indicator on the validation projects after each epoch of training, for the best considered hyperparameter configuration. The performance seems to keep increasing at the moment when the number of epochs of this evaluation has been reached and, therefore, the training has been stopped. Although better results could be probably achieved with additional epochs, the number of epochs is defined by the hyperparameter optimization at each of the 50 iterations and cannot be changed.

Table 5.6: Hyperparameter ranges for the tuning of a convolutional 2-dimensional neural network with future resource utilization

Hyperparameter	Min value	Value of the best configuration	Max value	Considered values
R_{max}	6	7	9	[6,7,8,9]
T	5	12	15	[5,6,...,14,15]
NoL_{NN_1}	4	5	6	[4,5,6]
NoF_{NN_1}	32	64	128	[32,64,128]
NoL_{NN_2}	4	5	6	[4,5,6]
NoF_{NN_2}	32	128	128	[32,64,128]
$NoL_{NN_{final}}$	4	5	6	[4,5,6]
$NoN_{NN_{final}}$	256	512	1024	[256,512,1024]
N_{epochs}	10	12	25	[10,11,...,24,25]
α	10^{-5}	$10^{-4.2}$	10^{-3}	[10^{-5},$10^{-4.9}$,...,$10^{-3.1}$,10^{-3}]
σ	0.5	0.67	0.9	[0.5,0.51,...,0.89,0.9]

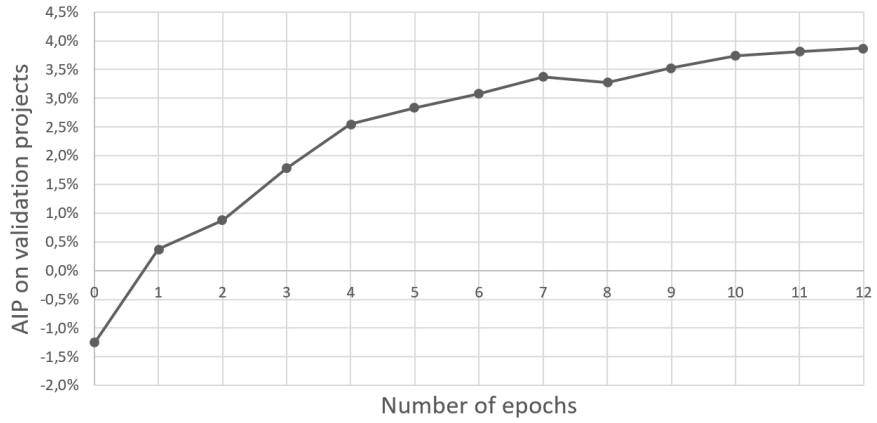

Figure 5.12: Performance of the convolutional 2-dimensional neural network with FRU.

The hyperparameter tuning with 50 sampled combinations for this neural network structure took 15.17 days.

5.3.7 Considerations on the results of the hyperparameter tuning

By analyzing results of the hyperparameter tuning for the six neural network structures (also available in full in Appendix A), it is possible to make many interesting observations.

First of all, the NN-CONV1D-FRU, followed by the NN-CONV2D-FRU, is the neural network structure that performed best after the tuning on validation projects (and after the training) as shown in Figure 5.13. Moreover, it seems that including the information about the future resource utilization improves the quality of the scheduling. These statements will be checked again on the test projects in the next sections.

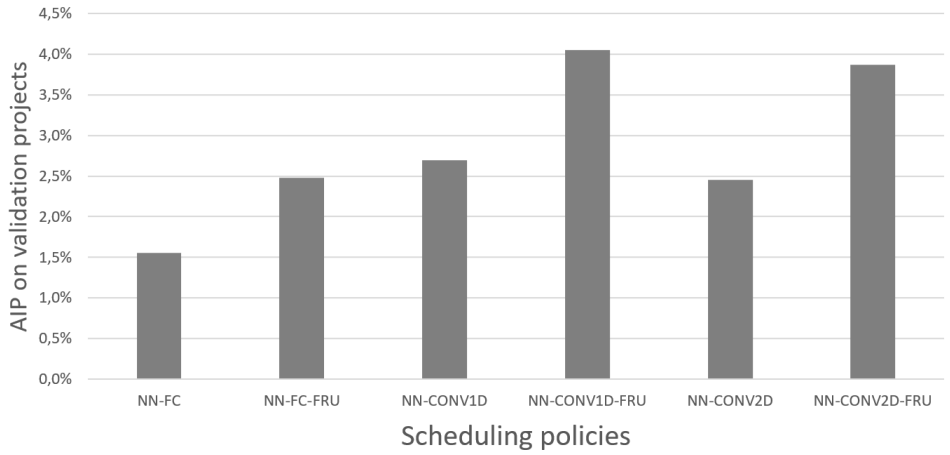

Figure 5.13: Performance comparison of neural network-based scheduling policies applied on the validation projects.

The Bayesian optimization seems to be effective since the best solution is found in the second half of the experiments in 5 out of 6 hyperparameter tunings (see bolded rows in the Tables of Appendix A). That means that, as more and more parameter configurations are explored, the algorithm identifies which regions of the solution space are the most promising ones and evaluate if it is worthy to exploit them.

The benefit of the hyperparameter tuning can be roughly estimated by considering also the average performance on the 50 tested hyperparameter combinations. Figure 5.14 and Table 5.7 shows that by using the hyperparameter tuning significantly higher performance can be achieved for all six neural network structures. The average increase of the AIP on the validation projects is +0.971% for the best neural network structure, i.e. the NN-CONV1D-FRU, and in general between +0.610% and +1.191% considering all six structures.

If the learning curve of some neural network structures with the best tested hyperparameter configuration is considered (e.g. Figure 5.12 and 5.4), the performance seems to keep increasing at the moment when the number of epochs of

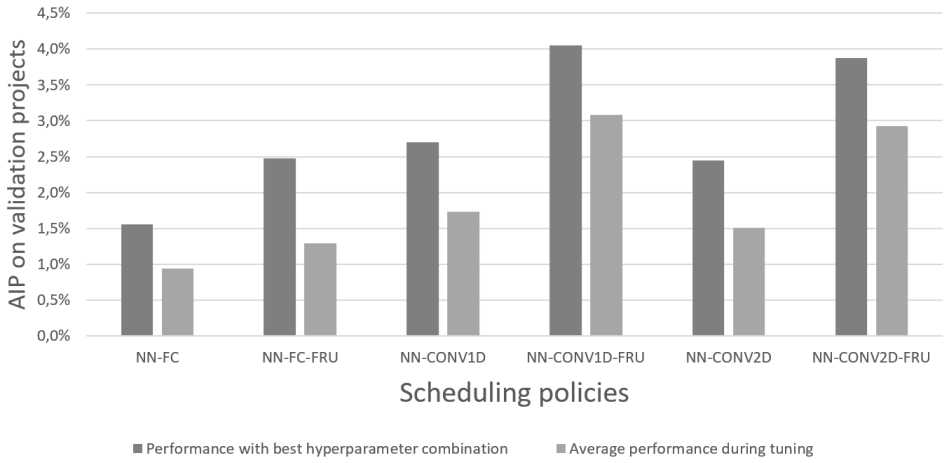

Figure 5.14: Performance comparison of tuned scheduling policies on the validation projects with average performance during the hyperparameter tuning.

Table 5.7: Numerical performance comparison of the tuned scheduling policies on the validation projects with average performance during the hyperparameter tuning

Scheduling policy	AIP with tuned hyperparameters [%]	AIP with centered hyperparameters [%]	AIP difference [%]
NN-FC	1.551	0.941	0.610
NN-FC-FRU	2.478	1.287	1.191
NN-CONV1D	2.697	1.731	0.966
NN-CONV1D-FRU	4.049	3.078	0.971
NN-CONV2D	2.450	1.509	0.941
NN-CONV2D-FRU	3.871	2.931	0.940

this evaluation has been reached and, therefore, the training has been stopped. Since the goal of this chapter is to find a good hyperparameter combination for each of the six neural network types, the number of epochs is defined prior to the beginning of the training and cannot be changed later, even though better results might be achieved with few additional epochs.

If one takes a look at the best hyperparameter combination in each of the six considered neural network structures, it is possible to observe the similarities explained in the following paragraphs. The following considerations are just qualitative, since the data does not come from a structured design of experiment but they are the result of a Bayesian optimization.

The chosen values for R_{max} are always either 6 or 7, while the predefined range is between 6 and 9. The Bayesian optimization is, then, suggesting that considering more "ready to start" activities in the start vector seems not to improve the scheduling policy after training. However, preliminary simulations have shown that, considering all the 50.000 generated projects, some situations may occur where up to 9 "ready to start" activities can be considered in the same decision point. One reason for that may be that these situations are so seldom that the increased model complexity is not justified. Figure 5.15 shows the sampled parameter combinations in a diagram with the AIP on the validation projects and the correspondent R_{max} value for the NN-CONV1D-FRU neural network structure.

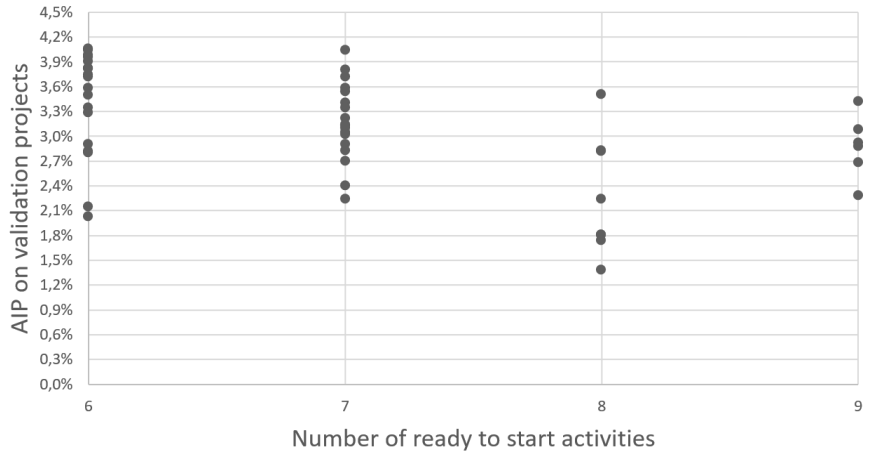

Figure 5.15: Sampled hyperparameter combinations in a R_{max} - AIP diagram for the validation projects and the NN-CONV1D-FRU.

The chosen values for the time horizon T are always between 11 and 14, while the predefined range is between 5 and 15. It seems not suitable to use lower values as less information would be considered as an input. Figure 5.16 shows the sampled parameter combinations in a diagram with the AIP on the validation projects and the correspondent T value for a NN-CONV1D-FRU neural network structure.

Different values have been considered for the number of layers NoL, the number of neurons per layer NoN and the number of filters per layer NoF too. For these hyperparameters, it is more difficult to make some qualitative considerations since no trend can be recognized in the graphs and different values of the same parameter are chosen for different neural network structures. The reason

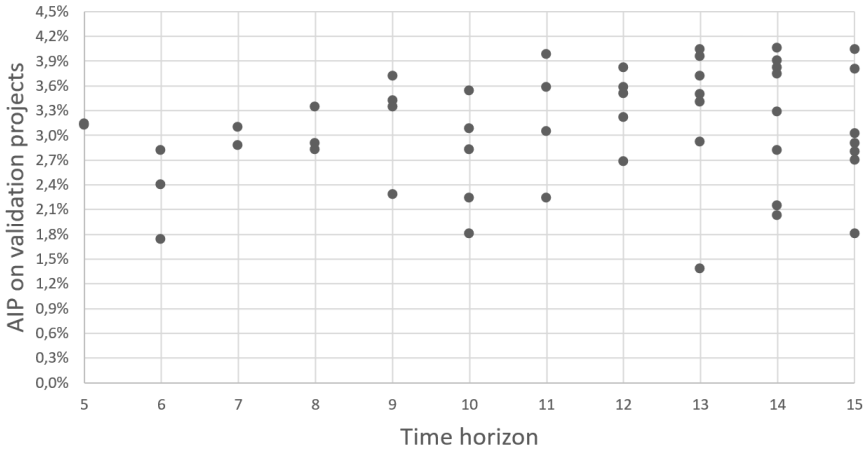

Figure 5.16: Sampled hyperparameter combinations in a T - AIP diagram for the validation projects and the NN-CONV1D-FRU.

could be that the values of the chosen ranges are all quite suitable to achieve a proper model capacity. Figure 5.17 shows the sampled parameter combinations in a diagram with the AIP on the validation projects and the correspondent $NoL_{NN_{final}}$ value for a NN-CONV1D-FRU neural network structure. In this case, it is possible to state that some values of $NoL_{NN_{final}}$ have been sampled more and, as a result, it is more likely that both very good and very bad values have been found for it. One possible suggestion for future investigation is to widen the parameter range to better identify possible trends.

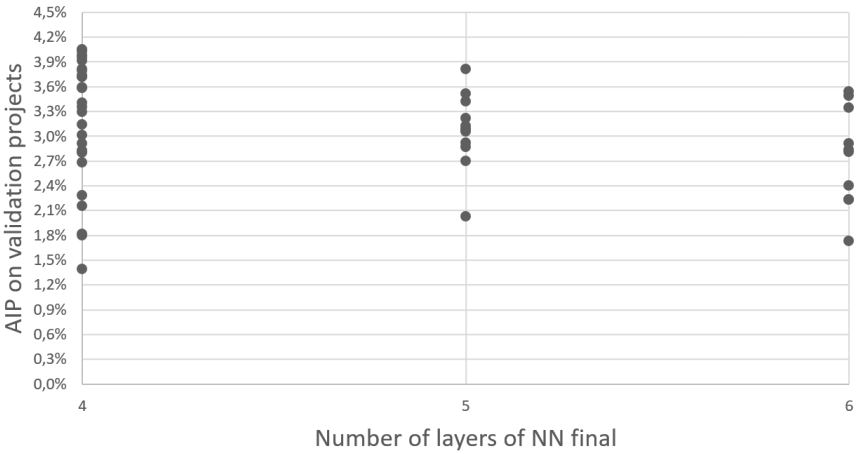

Figure 5.17: Sampled hyperparameter combinations in a $NoL_{NN_{final}}$ - AIP diagram for the validation projects and the NN-CONV1D-FRU.

The chosen values for the number of epochs N_{epochs} are always between 10 and 15, while the predefined range is between 10 and 25. Figure 5.18 shows, for example, the sampled parameter combinations in a diagram with the AIP on the validation projects and the correspondent T value for the NN-CONV1D-FRU neural network structure. In this diagram, it is possible to observe that all the best parameter configurations are concentrated on the left side of the graph. A similar trend can be also identified for the other neural network structures and that leads to the conclusion that for $N_{epochs} > 15$ the phenomenon of overfitting may become visible.

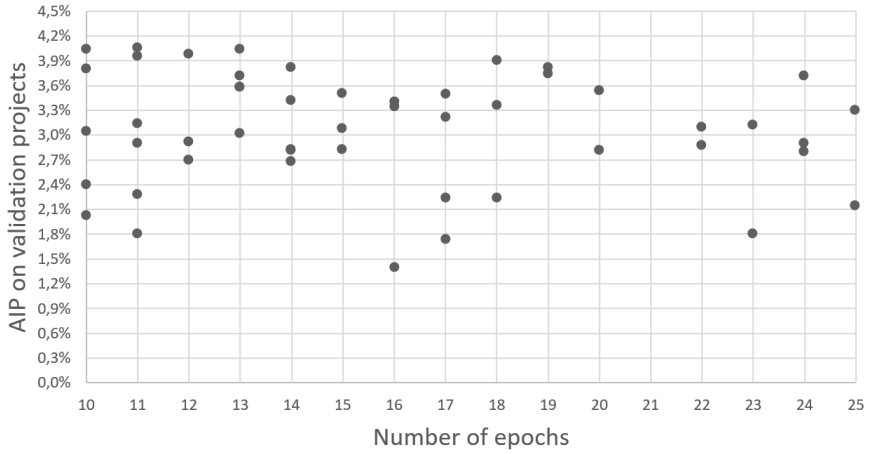

Figure 5.18: Sampled hyperparameter combinations in a N_{epochs} - AIP diagram for the validation projects and the NN-CONV1D-FRU.

The chosen values for the learning rate α are always between $10^{-3.8}$ and $10^{-4.5}$, while the predefined range is between 10^{-3} and 10^{-5}. Figure 5.19 shows the sampled parameter combinations in a diagram with the AIP on the validation projects and the correspondent α value for a NN-CONV1D-FRU neural network structure. In this diagram, it is clearly visible that the best performance can be achieved with a learning rate α in the middle of the considered range. It seems that, on one hand, a too small α let the neural network learn too slowly and, on the other hand, a too big α makes the gradient descent algorithm update the weight too reactively which leads to a worse training process. Similar diagrams and considerations on α can be done for the other neural network structures.

The chosen values for the dropout σ are always between 0.67 and 0.74, while the predefined range is between 0.5 and 0.9. Figure 5.20 shows the sampled parameter combinations in a diagram with the AIP on the validation projects and the

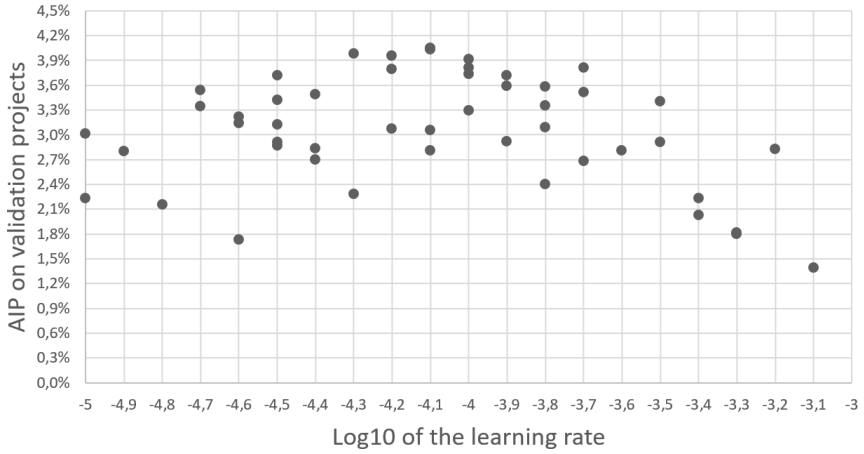

Figure 5.19: Sampled hyperparameter combinations in a α - AIP diagram for the validation projects and the NN-CONV1D-FRU.

correspondent σ value for a NN-CONV1D-FRU neural network structure. In this diagram, it is clearly visible that the best performance can be achieved with dropout value σ in the middle of the considered range. Values around 70% are quite different than the ones suggested by some literature contributions like, for example, Hinton, Srivastava, Krizhevsky, Sutskever, and Salakhutdinov (2012) where values of 50% are used. Similar diagrams and considerations on σ can be done for the other neural network structures.

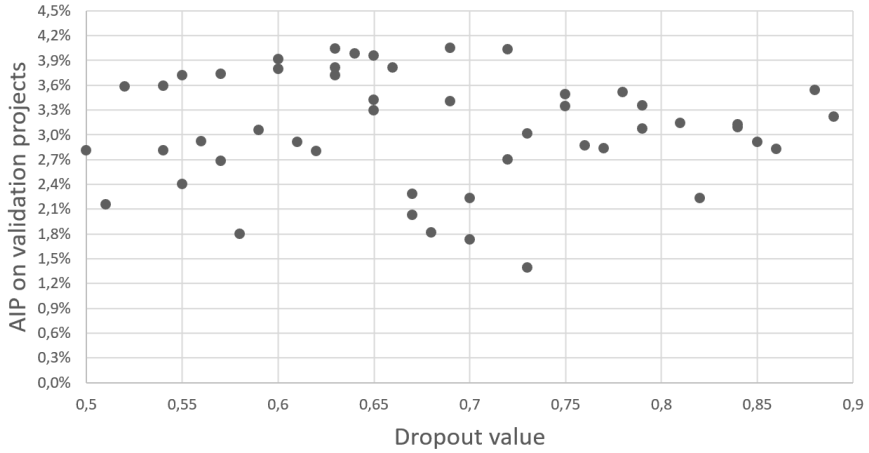

Figure 5.20: Sampled hyperparameter combinations in a σ - AIP diagram for the validation projects and the NN-CONV1D-FRU.

5.4 Benchmark on the test projects

The goal of the previous step, i.e. the hyperparameter tuning, was to find good hyperparameters for the six considered neural network types in order not to compromise their performances. Once good values for the hyperparameters have been found, it is possible to move on to the next step and test the six tuned neural networks against each other and against other considered reactive scheduling policies on the unseen test projects.

Figure 5.21 and Table 5.8 show the performances of all considered policies in term of AIP on the test projects. The random policy RAN always has an AIP of zero, since the AIP is defined as a performance gap from the RAN policy itself. Assuming that no policy worse than random is considered, the AIP_{RAN} represents the lower-bound for the AIP indicator. On the other hand, the LB-10000 policy represents its upper-bound. The name LB, i.e. lower-bound, comes from the convention in the literature where the better the scheduling algorithm the lower the performance indicator linked to the project makespan and not vice versa as for the AIP. As long as all activity durations are deterministic and known, the upper-bound is reachable and the LB-10000 performs best. However, the assumption of this thesis is that, even though the problem is modeled with deterministic times, a certain level of uncertainty is expected for the activity durations and, as a result, this upper-bound is considered as unreachable.

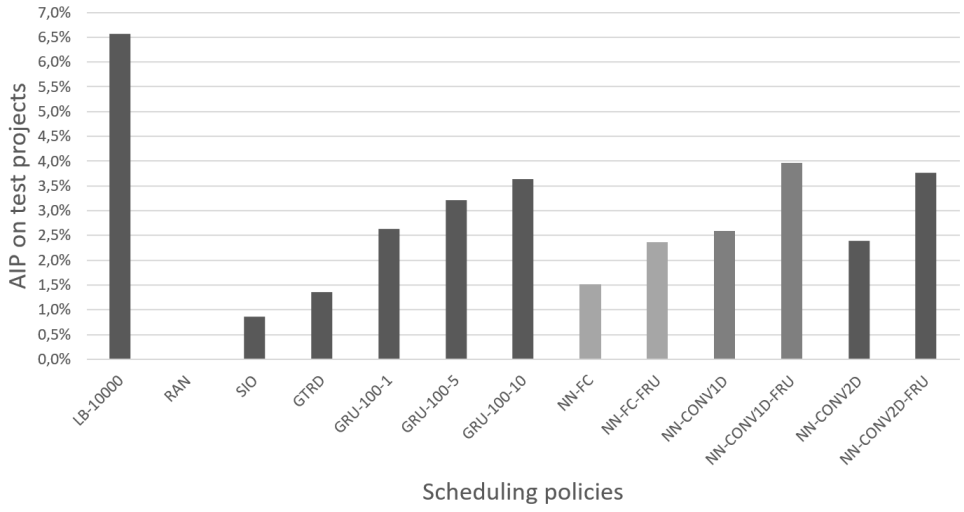

Figure 5.21: Performance comparison among the considered scheduling policies applied on the test projects.

Table 5.8: AIP performance indicators of the considered scheduling policies on the training, validation and test policies

Scheduling policy	AIP training projects [%]	AIP validation projects [%]	AIP test projects [%]	Average decision time [*seconds*]
LB-10000	6.543	6.507	6.567	1.7
RAN	0.000	0.000	0.000	$2.5 \cdot 10^{-6}$
SIO	0.855	0.895	0.863	$3.5 \cdot 10^{-6}$
GTRD	1.386	1.397	1.362	$2.3 \cdot 10^{-5}$
GRU-100-1	2.640	2.675	2.636	$1.2 \cdot 10^{-3}$
GRU-100-5	3.234	3.245	3.212	$1.9 \cdot 10^{-2}$
GRU-100-10	3.621	3.617	3.637	$5.6 \cdot 10^{-2}$
NN-FC	1.640	1.551	1.510	$3.8 \cdot 10^{-4}$
NN-FC-FRU	2.650	2.478	2.370	$6.6 \cdot 10^{-4}$
NN-CONV1D	2.828	2.697	2.587	$4.7 \cdot 10^{-4}$
NN-CONV1D-FRU	4.177	4.049	3.964	$4.6 \cdot 10^{-4}$
NN-CONV2D	2.558	2.450	2.398	$9.4 \cdot 10^{-4}$
NN-CONV2D-FRU	3.983	3.871	3.767	$1.1 \cdot 10^{-3}$

Among the considered heuristics, the ones based on the GRU-S-T algorithm have achieved the best results and the larger the parameter T, the higher the performance. As this class of heuristics results the best one, it is further evaluated in Section 5.5.1 with more value combinations for S and T against the best policy based on neural networks.

Considering the neural network-based policies, it is possible to see that the ones including the future resource utilization in the input information achieved an AIP which is between 0.86% and 1.376% higher than the correspondent policies without this information. The NN-CONV1D policy achieved the best performance among all the considered ones also on the test projects confirming the results on the validation projects.

If one compares the results of the neural network-based policies also on the training and validation projects, it is possible to notice a systematic gap between the performance on the training set and the validation set as shown in Figure 5.22. This gap is expected to tend to zero as the number of projects goes to infinite. In this thesis, it has an average value of 0.123%. Even smaller is

the gap between the performance on the validation projects and on the test projects which is 0.083% on average.

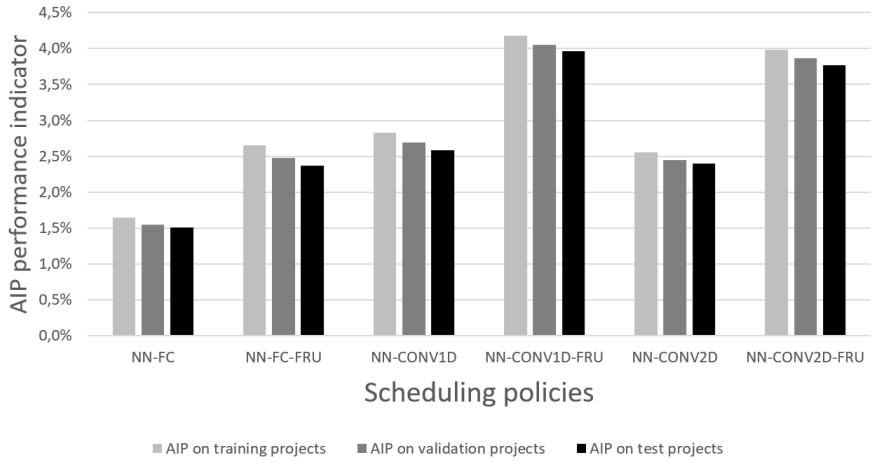

Figure 5.22: Performance comparison of neural network-based scheduling policies applied on the training, validation and test projects.

Figure 5.23 and Table 5.8 shows the decision times for each scheduling policy, i.e. the time that the decision algorithm needs to compute the priority values for the "ready to start" activities. It is important to notice that the decision is represented on a logarithmic scale and is strongly related to the hardware and software that was introduced in Section 5.1.

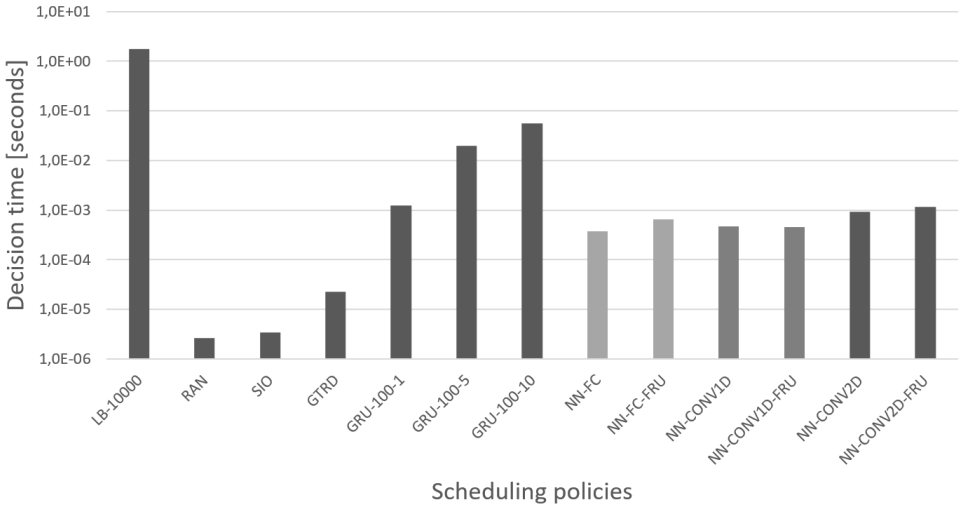

Figure 5.23: Average decision times of considered scheduling policies.

The slowest algorithm is the LB-10000 since it is based on a brute-force Monte Carlo simulation with 10.000 runs and random decisions. However, as mentioned in the previous paragraphs, this policy is not considered since it is strongly based on certain information about the activity durations. Considering the remaining heuristics, it is possible to notice that the fastest ones are the ones associated with the worst AIP performance. In fact, the GRU-100-10 scheduling policy, which is the most competitive one against the neural network-based policies, has an average decision time of $5.6 \cdot 10^{-2}$ seconds which is approximately 122 times higher than the one of the NN-CONV1D-FRU policy.

5.5 Further considerations

For the sake of simplification, the default values of a big number of parameters have been defined à priori by the author. The following subsections aim to assess if these predefined values have been correctly chosen.

The previous step, i.e. the hyperparameter tuning, is not only used to determine the best values for the hyperparameters but also to determine which neural network structure seems to perform at best. In fact, if the goal is to design a neural network that can effectively schedule the activities in the resource constrained project scheduling problem, the neural network type could also be considered as a hyperparameter. In the last step of the proposed methodology, i.e. the benchmark on the test projects, the neural network that obtained the best performance in terms of AIP on the validation projects during the hyperparameter tuning among all six considered neural network types is deployed for final evaluation on the test projects.

Among the 300 evaluations of different neural network types with different hyperparameters, the NN-CONV1D-FRU structure with the best hyperparameter configuration introduced in Table 5.4 obtained the best performance in term of AIP. Only this neural network will be used for the further considerations of the following sections.

5.5.1 Parameters of the GRU heuristics

Up to now, only three scheduling policies of the class GRU-S-T have been considered, i.e. the GRU-100-1, the GRU-100-5 and the GRU-100-10. However,

much more parameter combinations could be considered and it is possible to evaluate whether and for which combination of S and T this policy class outperforms the NN-CONV1D-FRU scheduling policy along with the corresponding decision time.

Figure 5.24 shows how the performance indicator AIP on the test projects changes for different combinations of S and T. The best performance can be achieved with the highest considered value for both S and T. This parameter combination even outperforms the performance of the NN-CONV1D-FRU scheduling strategy by 0.101% but it relies on a longer time horizon, which may be associated with more uncertainty and implies a decision time 3370 times bigger (see Figure 5.25).

If only one of the two parameter is too low, the entire AIP indicator gets compromised. As a result, one could state that, in order to increase the AIP indicator, both the number of schedules S and the time horizon T should be increased simultaneously.

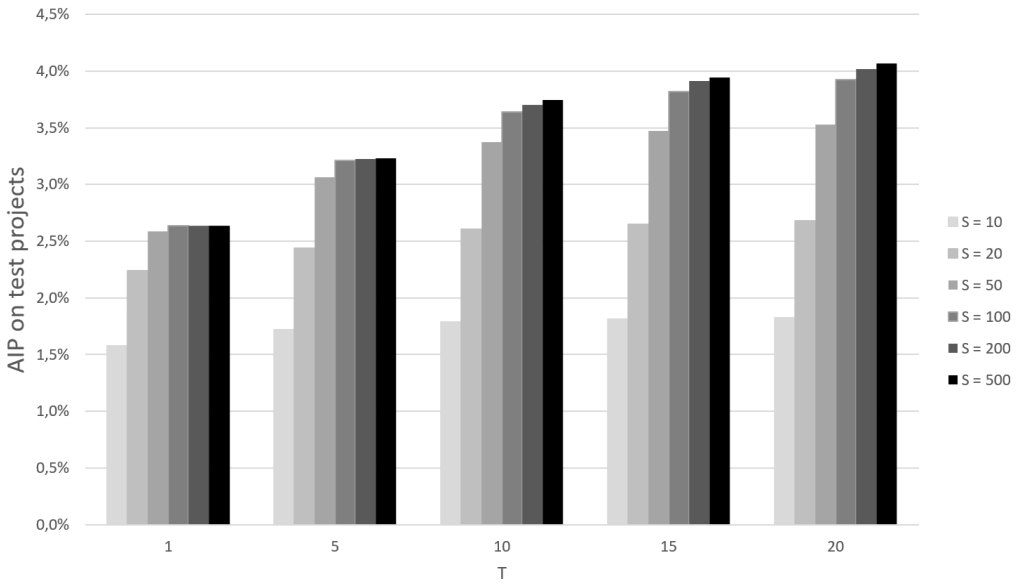

Figure 5.24: AIP performance indicator for the GRU scheduling policy on the test projects as a function of parameter S and T.

Figure 5.25 shows how the average decision time on test projects changes for different combinations of S and T. The considerations that can be done on this diagram are similar to the ones for Figure 5.24. In general, it is possible to state that the more schedules are evaluated and the longer the schedule horizon

of a GRU-S-T scheduling policy, the higher the correspondent performance but also the decision time.

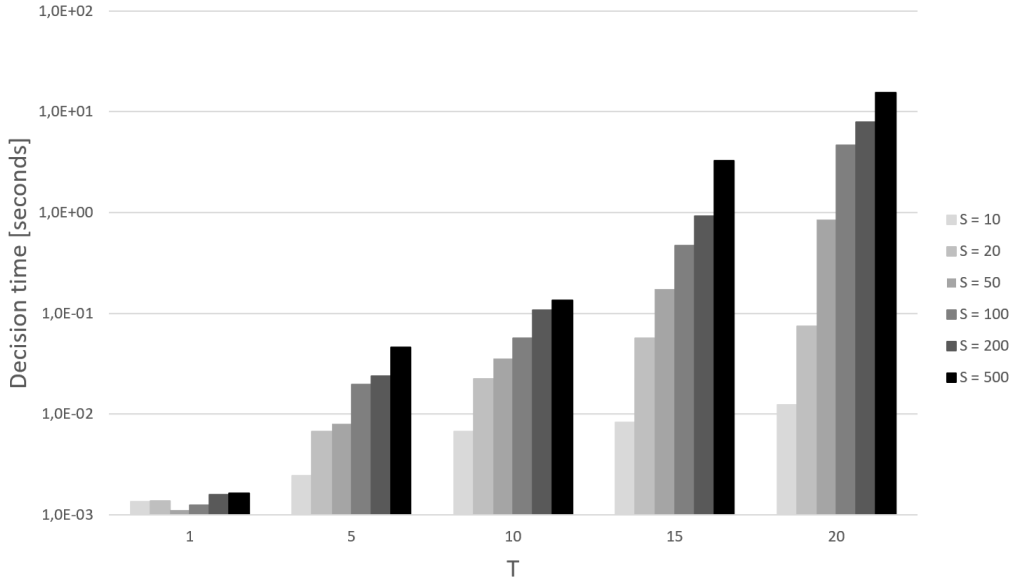

Figure 5.25: Average decision times for the GRU scheduling policy on the test projects as a function of parameter S and T.

The assumed value of 100 for S and 10 for T seems to be a reasonable compromise between performance and decision time which is only 120 times higher than the one for the NN-CONV1D-FRU scheduling policy. The two parameter combinations which perform better than the tuned NN-CONV1D-FRU policy, are associated with decision times of 7.93 and 15.4 seconds which may be acceptable for same application but unacceptable for others, like, for example, the use in a simulation environment where it may be necessary to take many decisions per second or if the hardware is less performing than the one used to generate the presented results (see section 5.1).

5.5.2 Number of runs to compute the AIP upper bound

The LB-N scheduling policy has been used through this thesis to estimate an upper bound for the AIP indicator. A value of 10.000 for the parameter N has been assumed from the beginning and this section aims to justify this choice. Figure 5.26 shows how the AIP indicator on the test projects changes for different values of N. In particular, the value of AIP increases together with the

value of N and reaches its highest value with N equal to 100.000. However, the performance increment decreases rapidly and it seems to reach a saturation point at N equal to 10.000.

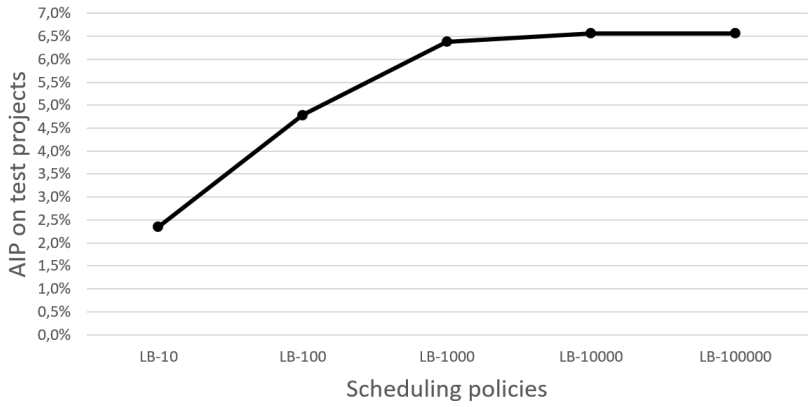

Figure 5.26: AIP performance indicator for the LB-N scheduling policy on the test projects as a function of parameter N.

On the other hand, as Figure 5.27 shows, the average computing time per project increases quite linearly with the parameter N. As a result, it is reasonable to choose a value of 10.000 for N since it is still feasible from the computational point of view and seems to approximate well the upper bound for the AIP.

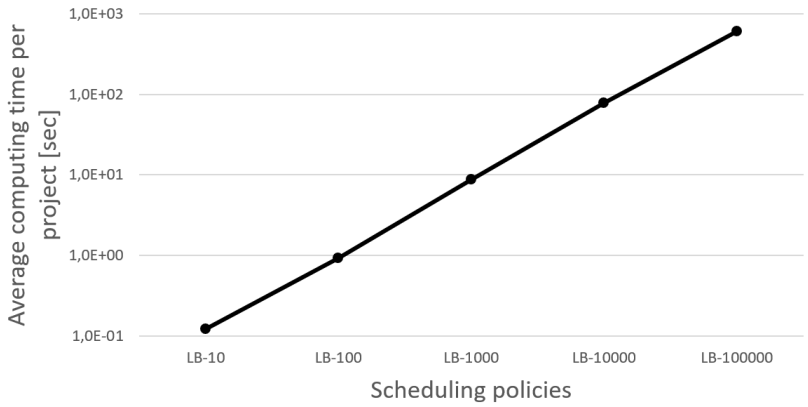

Figure 5.27: Average computing time per project to find the lower bound with the LB-N scheduling policy on the test projects as a function of parameter N.

5.5.3 Number of runs to create training data

As mentioned in Section 4.5.1, the training data are generated by performing 10.000 runs with a RAN scheduling policy and by selecting the state-action pairs of the best run. Of course, the more random simulation runs are done, the more likely is to find a run with the lowest possible makespan or at least with a very low one, which is associated with very good state-action pairs. The better the state-action pairs, the better the performance of the neural network after the training.

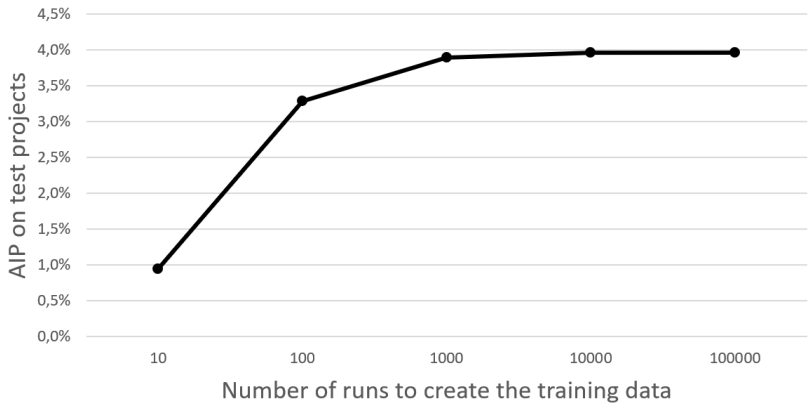

Figure 5.28: AIP performance indicator for the tuned NN-CONV1D-FRU scheduling policy on the test projects as a function of number of runs to create the training data.

Figure 5.28 shows the AIP performance indicator for the tuned NN-CONV1D-FRU scheduling policy on the test projects for different numbers of runs used to create the training data. It is possible to observe that the AIP increases together with the number of runs but it reaches a saturation point when 10.000 runs are used. As a result, it is correct to use this value.

This value is also feasible from the computational point of view as shown in Figure 5.29.

5.5.4 Number of training projects to create training data

The training data for the previous results have been generated using an à priori defined number of training projects equal to 40.000. In general, the more training projects are used, the more different state-action pairs are generated, the spectrum of considered decisions gets wider and, as a result, the risk of overfitting decreases.

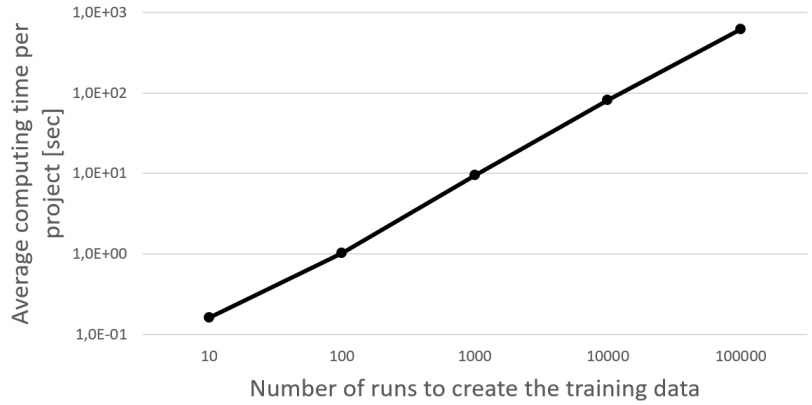

Figure 5.29: Average computing time per project to create the training data as a function of number of runs.

Figure 5.30: AIP performance indicator for the tuned NN-CONV1D-FRU scheduling policy on the test projects as a function of number of training projects.

Figure 5.30 shows how the AIP performance indicator changes if the number of training projects varies. It seems that the AIP on the test projects reaches a saturation point for a number of training projects bigger than 30.000. As a result, 40.000 seems a reasonable value also considering the computing time as shown in Figure 5.31. It is important to notice that, as the number of training projects increases, new projects are added in the evaluation. These new projects may be easier or harder to schedule and some small deviations may become visible even if the saturation point is reached.

Figure 5.31: Average computing time per project to create the training data as a function of number of training projects.

5.5.5 Use of the resource and activity conversion vector

As mentioned in Section 4.3.1, both a resource and an activity conversion is used to represent similar situations with a similar input vector and matrix. This à priori decision is justified in the following paragraph.

Figure 5.32 shows how the AIP of the tuned NN-CONV1D-FRU on the test projects would decrease if these two conversions were not used.

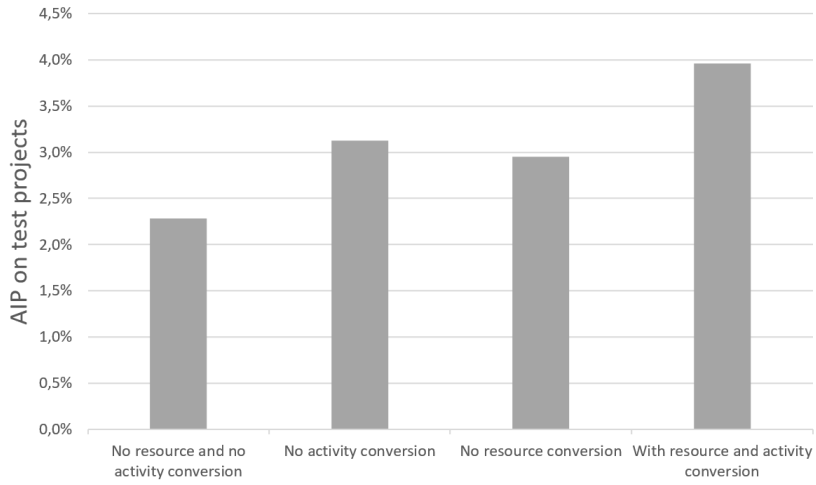

Figure 5.32: AIP performance indicator for the tuned NN-CONV1D-FRU scheduling policy on the test projects with and without resource and activity conversion.

In particular, the worst performance is obtained when both conversions are not used, while the use of only one of the two conversions leads to a lower

performance degradation. Lastly, the resource conversion seems to be more important than the activity conversion since a lower performance is obtained when only the first one is left out.

5.5.6 Use of the input normalization

Ketkar and S. (2017) suggests to normalize the input values in such a way that their absolute value is between 0 and 1. Similarly, also in this thesis, the values of the input vector $V_{ReadyToStartActivities}$ and of the input matrix $M_{FutureResourceUtilization}$ have been normalized using either the duration rescale factor RF or the total available resource quantity R_k. This assures that most of the absolute values are included between 0 and 1 and drastically reduces the others.

Figure 5.33 shows how the AIP of the tuned NN-CONV1D-FRU on the test projects would decrease if the normalization of the input vector and matrix was not used. In particular, it can be noticed that the worst performance is obtained when both normalizations are not used, while the normalization of only one of the two pieces of input information leads to a lower performance degradation. The normalization of the input vector seems to be more important than the normalization of the input matrix since a lower performance is obtained when only the first one is not used.

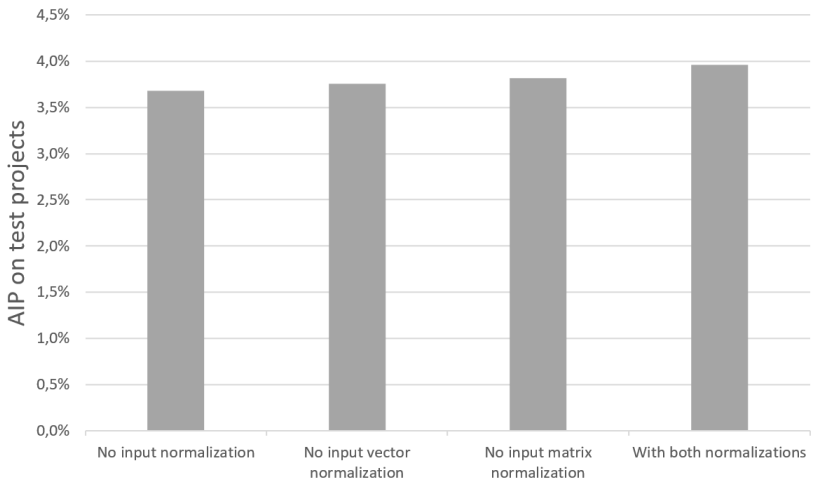

Figure 5.33: AIP performance indicator for the tuned NN-CONV1D-FRU scheduling policy on the test projects with and without the normalization of the input vector and matrix.

The impact of the input normalization (0.285% performance degradation) is, however, much less than the impact of the resource and activity conversion (1.681% performance degradation).

6 Sensitivity analysis

The potential benefits of artificial intelligence are huge, so are the dangers.
-D. Waters

So far, no uncertainty in the characteristics of the activity sequence and in the resource availability has been considered. In reality, the à priori defined features of the problem may be just estimated by a group of experts or with preexisting historical data and they may not consider some unexpected events that can change them. As a consequence, they may deviate from the real project characteristics that become known only after the project execution. For example, some resources may become temporally unavailable, some activities may require more or less resources or time and so on.

This chapter presents a sensitivity analysis done considering random deviations of the activity durations. First of all, the scope and the characteristics of the experiments are described and, then, the results are presented.

6.1 Scope of the sensitivity analysis

The considered scheduling policies take decisions by considering the assumed activity durations. In reality, the activities may take more or less time than expected and that may lead to a decrease of the AIP performances. The scope of this chapter is to determine whether and how much the performance of the different scheduling policies decreases if this kind of deviations occurs.

In particular, 11 different levels of uncertainties for the activity durations are considered. The real activity duration of a generic activity j is no more considered as a deterministic value, but it is characterized by a uniformly distributed random variable denoted as $\hat{d}_j$ and centered in the assumed value d_j. Each

uncertainty level is characterized by the extreme values of the uniform distribution as shown in Table 6.1.

Table 6.1: Uncertainty levels of the sensitivity analysis

Uncertainty level	Min.	$E[\hat{d}_j]$	Max
±0%	d_j	d_j	d_j
±10%	$0.9 \cdot d_j$	d_j	$1.1 \cdot d_j$
±20%	$0.8 \cdot d_j$	d_j	$1.2 \cdot d_j$
±30%	$0.7 \cdot d_j$	d_j	$1.3 \cdot d_j$
±40%	$0.6 \cdot d_j$	d_j	$1.4 \cdot d_j$
±50%	$0.5 \cdot d_j$	d_j	$1.5 \cdot d_j$
±60%	$0.4 \cdot d_j$	d_j	$1.6 \cdot d_j$
±70%	$0.3 \cdot d_j$	d_j	$1.7 \cdot d_j$
±80%	$0.2 \cdot d_j$	d_j	$1.8 \cdot d_j$
±90%	$0.1 \cdot d_j$	d_j	$1.9 \cdot d_j$
±100%	0	d_j	$2 \cdot d_j$

For what concerns the neural network-based scheduling policies, only the one based on the best neural network structure is considered for the results in the following section, i.e. the NN-CONV1D-FRU, while all heuristic scheduling algorithms used in Section 5.4 are considered for the sensitivity analysis.

6.2 Results of the sensitivity analysis

Considering the uncertainty levels defined in the previous section, it is now possible to compute the correspondent AIP performances for all the considered scheduling policies.

As for chapter 5, the confidence intervals for the AIP values are intentionally omitted in Figure 6.1. In this case the activity durations are affected by random deviations which add an additional source of variability. Despite of this, thanks to the large number of project instances and the large number of simulation runs (10.000 for RAN and 100 for the other scheduling policies) used to estimate the AIP indicators on each project instance, the variance is very small compared to the scale of the diagrams also in this case. As a result, the confidence intervals would be hardly visible and they have been omitted

for clarity purposes. Considering 95% confidence intervals created with 5 samples, it is possible to state that all confidence intervals for the AIP indicator of the scheduling policies are less or equal than 0.0528% and increase as the uncertainty on the activity duration increases.

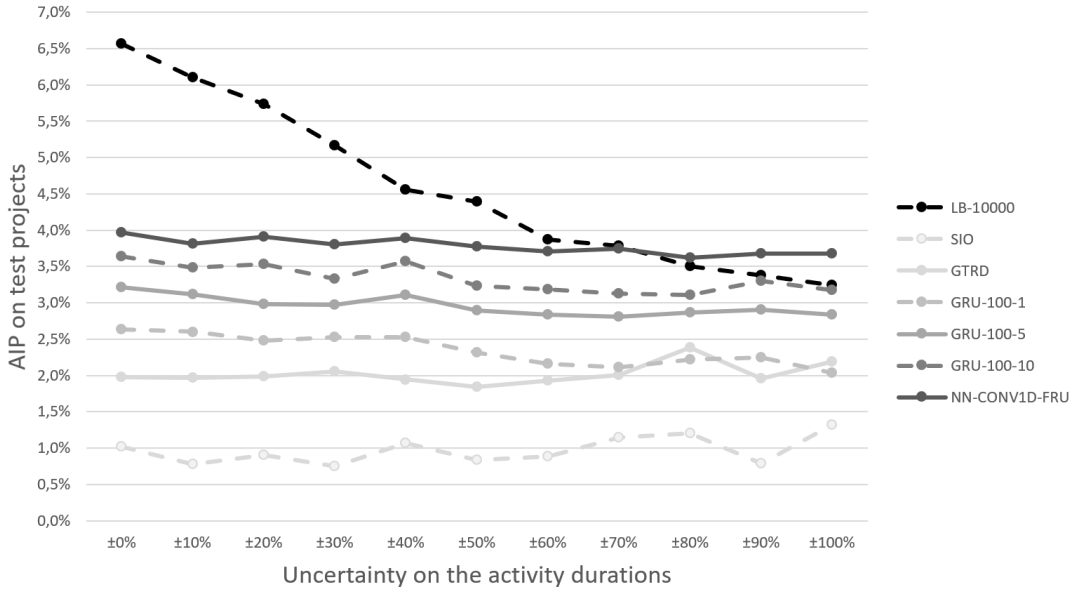

Figure 6.1: AIP performance indicator on the test projects for different scheduling policies and different levels of uncertainties.

Figure 6.1 shows how the AIP indicator on the test projects varies as a function of the uncertainty level for all considered scheduling policies. In general, it is possible to say that, for most of the scheduling policies, the performance slowly decreases as the level of uncertainty increases.

For what concerns the policies SIO and GTRD, no visible ascending or descending trend can be identified. Instead, it seems that there is a random fluctuation. The reason could be that, since their algorithms are not based on creating schedules in a predefined time horizon, they are not sensible to variations of the activity durations.

The LB-10000 policy is the one that faces the strongest performance degradation as the uncertainty level increases. One possible reason is that its algorithm is based on the generation of 10.000 schedules considering all activities that have not been completed yet, i.e. generally with a very long time horizon. As the uncertainty level increases, the planned starting times s_j of the best schedule, may become unfeasible. The AIP performance indicator of the LB-10000 policy

gets even worse than the one of NN-CONV1D-FRU for high level of uncertainties and that seems to suggest that considering long time horizons with high levels of uncertainty may even compromise the performances of a scheduling policy as the decision is based on unreliable information.

For what concerns the NN-CONV1D-FRU scheduling policy, it is possible to observe a similar performance degradation as for the policies of the GRU-S-T class. As expected, the highest value for AIP is obtained for an uncertainty level of $\pm 0\%$ and is equal to 3.964 %, while the lowest, which is obtained for an uncertainty level of $\pm 80\%$, is 3.623 %. The performance degradation can be than quantified by computing the difference between these two values which is 0.341 %. Compared to the initial AIP value of 3.964 % and considering the high uncertainty level, a performance degradation of 0.341 %, which is approximately 8.6 % of the initial value, is quite a good value and it is possible to conclude that the NN-CONV1D-FRU scheduling policy has good perfomances even if the activity durations have high uncertainty levels.

7 Conclusions

The great paradox of automation
is that the desire to eliminate
human labor always generates
new tasks for humans.
-M. L. Gray

The aim of this final chapter is to summarize the results and findings of the thesis in Section 7.1 and to give an outlook on possible future research areas related to the presented topic in Section 7.2.

7.1 Summary

In this thesis, a new approach for the Resource-Constrained Project Scheduling Problem (RCPSP) has been proposed. In contrast to most of the literature contributions, where exact, heuristic and meta-heuristic methods are used, the author introduced a methodology based on artificial neural networks trained with supervised learning.

The proposed scheduling policy is reactive. In fact, it does not determine the starting time for each activity at the beginning of the project but it only defines which activities should be immediately started at each decision point, i.e. at the project's beginning and every time one or more activities are finished and new resources are released. The decision-making process uses artificial neural networks to process the numerical input information, which describes the current state of the activity sequence by means of a "ready to start" activity vector and a future resource utilization matrix, and to compute the priority values for the "ready to start" activities. After that, the possibility to start the "ready to start" activities is evaluated starting from the ones with the highest priority values.

Six neural network structures have been proposed and compared. Some of them use both the "ready to start" activities vector and the future resource utilization matrix, while others use only the first one. The results have shown that the first group of neural networks performs much better than the second one and that a 1-dimensional convolutional neural network is the most suitable neural network type to process the "ready to start" activities vector. As for image recognition, fully connected neural networks seem to be not very suitable for this task.

The hyperparameters of the neural network structures have been tuned with a Bayesian optimization whose goal was to maximize a performance indicator on the validation projects after the training. The hyperparemeter tuning helped to significantly improve the performances and to identify the most promising combinations. The most important hyperparameters seem to be the learning rate and the number of epochs. However, a too low time horizon may lead to poor performances as well. Within the considered ranges for the number of layers and the number neurons or filters per layer, it was difficult to recognize a clear trend and different values have been used for the six neural network structures.

For what concerns the convolutional neural networks, many concepts and guidelines from the field of object recognition have been used to roughly define their structures and hyperparameter ranges. The results of the hyperparameter tuning showed that some common practices coming from this field of research are not applicable in the RCPSP problem. For instance, the hyperparameter tuning always suggested to use a dropout value between 67 % and 74 %, while values below 50 % are generally suggested in the field of object recognition. Moreover, no max pooling could be used in the convolutional layers due to the small input vector and matrix.

The best neural network-based reactive scheduling policy, denoted as NN-CONV1D-FRU, has been tested against other reactive heuristic algorithms and a benchmark has been created. The results have been shown that this scheduling policy outperforms most of the heuristics. Only three heuristics achieved better performances. The first one is the so-called LB-10000 which, however, strongly relies on the estimation of the activity durations and implies decision times approximately 3.700 times bigger. The second and the third one are the GRU-200-20 and GRU-500-20 which are slightly better than the NN-CONV1D-FRU policy but they imply decision times, respectively, 17.390 and 33.771 times higher.

Due to the large number of parameters influencing the results, only some of them have been included in the hyperparameter tuning. For what concerns the remaining ones, some reasonable default values have been defined after some preliminary tests. After the training of the neural networks and after the hyperparameter tuning, the quality of these default parameter values have been checked again considering the NN-CONV1D-FRU scheduling policy with tuned hyperparameters as a baseline. This validation has shown some interesting results. For example, that 40.000 training projects and 10.000 random simulation runs for each one are enough to create good training data or that the preprocessing of the input data for the neural networks with the resource and activity conversion and the input normalization provide significant benefits.

One of the biggest challenges in scheduling activities with a neural network was to cope with a variable number of "ready to start" activities. That would have implied a variable size of the input and output vector which is not possible to handle with a neural network. The choice of defining a maximum number of considered "ready to start" activities and setting to zero the positions of the input vector that are not used solved this problem effectively.

7.2 Outlook

Although many aspects of the proposed methodology have been considered, there are still many research areas related to this topic that could be further investigated.

In this thesis, the activity durations have been considered as known and deterministic and the training has been done using training data based on this assumption. After that, a sensitivity analysis has been carried out to assess what happen if the activity duration are different in reality. Under certain circumstances, the activity durations may be stochastic and with a known probability distribution, for example, if a large amount of historical data is available. In this case, no sensitivity analysis is required and the training data can be generated considering the so-called Stochastic-RCPSP.

Another interesting research field for future work is how to define a state matrix that does not imply information loss about the topology of the activity sequence. In the proposed approach, this information is merged into a future resource utilization matrix of a predefined shape which is built without consid-

ering the resource constraints. That implies an information loss and, probably, a consequent performance loss. A recently proposed neural network type called graph convolutional neural network is able to directly extract features and hence information from graph structured data. A variant of this network has already been successfully applied on the Traveling Salesman Problem Kool, Van Hoof, and Welling (2018). Since every RCPSP instance represents an acyclic graph, this approach could also be beneficiary for this kind of problem as well.

The proposed methodology is based on machine learning and, in particular, on supervised learning. The supervised learning approach has been chosen over reinforcement learning by author due to the fact that in the latter the neural network is supposed to learn not only from exemplary state-action pairs but basically from the experience at every decision point with both positive and negative rewards. After some preliminary tests, the supervised learning approach seemed to be much faster and efficient and it has been chosen for this reason. However, it is not possible to exclude that a reinforcement learning approach can achieve better results even if the training process seems to be much more time consuming. The two approaches could be even combined as it has been done in Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, and Lanctot (2016) where supervised learning has been used to achieve a human level of playing skills, while reinforcement learning has been used further improve the algorithm and to make it able to beat the former world champion.

The generality is a key characteristic of a machine learning approach since it makes the same trained model able to solve many different situations. The neural networks presented in this thesis have been trained to solve a large variety of RCPSP problems, where the duration, the resource consumption, the number of activities and the sequence topology can vary under certain rules characterizing the chosen project class. However, other characteristics, for example the total number of available units for each resource type, were kept fixed. As a result, if these characteristics change, the performance of the neural networks is expected to decrease since they have been trained to solve a different class of problems and a new training with new training data is required. As an extension of the methodology presented in this thesis, a neural network able to solve RCPSP problems with a variable number of available resources could be developed in the future. However, the enhanced generality results in a more

complex task, which may require deeper and more complex neural networks to obtain comparable performances.

Despite the great potential of the application of neural networks in planning and scheduling problems, few contributions could be found in the literature. Due to the recent progresses in machine-learning, it is expected that similar methodologies can provide good results in this kind of problems and, hopefully, many findings and results of this thesis can be used as a start for these new research areas.

Nomenclature

$\bar{\mathcal{H}}$	Set of fixed hyperparameters
$\mathbb{F}$	Set of the "future" activities at the decision point t_d
$\mathbb{I}$	Set of the "idle" activities at the decision point t_d
$\mathbb{J}$	Set of the project's activities
$\mathbb{K}$	Set of renewable resource types
$\mathbb{P}$	Set of the "in progress" activities at the decision point t_d
$\mathbb{R}$	Set of the "ready to start" activities at the decision point t_d
$\mathcal{H}$	Set of hyperparameters to tune
Π	Scheduling policy
A_k	Current available quantity of resource units for the k^{th} resource type
$A_{k,t}$	Current available quantity of units for the k^{th} resource type at time t
ACV_j	Activity criticality value of activity j
AIP_{Π}	Average improvement percentage from the random policy RAN for the scheduling policy Π
d_j	Time required to complete the $j^{th}activity$
$M_{k,t}$	Element of the future resource utilization matrix $M_{FutureResourceUtilization}$
MS_p	Makespan of the project p
NN_1	First neural network to process the information about the "ready to start" activities and the currently available resources in the six considered neural network structures

NN_2	First neural network to process the information about the future resource utilization in the six considered neural network structures
NN_{final}	Last neural network to process the input information in the six considered neural network structures. Its output is also the output of the entire neural network model
$NRU_{j,k}$	Normalized resource utilization for activity j and resource type k
P	Number of projects in a project library
PV_j	Priority value for a "ready to start" activity
R_k	Total quantity of resource units for the k^{th} resource type
$r_{j,k}$	Number of required resource units of type k for the j^{th} activity
R_{max}	Maximum number of "ready to start" activities considered in the ready to start activity state vector
RCV_k	Resource criticality value of resource type k
RU_s	Resource utilization of a schedule s
s_j	Starting time in the considered schedule for the j^{th} activity
ACO	Ant-colony optimization
AoA	activity-on-the-arc
AoN	activity-on-the-node
GA	Genetic algorithm
GRU	Greatest resource utilization (scheduling policy)
GRU-S-T	Greatest resource utilization considering S schedules in the time horizon T (scheduling policy)
GTRD	Greatest total resource demand (scheduling policy)
J120	Project library with 120 activities per project generated with the ProGen project generator tool
J30	Project library with 30 activities per project generated with the ProGen project generator tool

J60	Project library with 60 activities per project generated with the ProGen project generator tool
LB	Lower bound scheduling policy
LB-CPM	Lower bound critical path method
LB-N	Lower bound scheduling policy considering N random simulation runs
NN-CONV1D	Convolutional 1-dimensional neural network without considering the future resource utilization (scheduling policy)
NN-CONV1D-FRU	Convolutional 1-dimensional neural network considering the future resource utilization (scheduling policy)
NN-CONV2D	Convolutional 2-dimensional neural network without considering the future resource utilization (scheduling policy)
NN-CONV2D-FRU	Convolutional 2-dimensional neural network considering the future resource utilization (scheduling policy)
NN-FC	Fully connected neural network without considering the future resource utilization (scheduling policy)
NN-FC-FRU	Fully connected neural network considering the future resource utilization (scheduling policy)
PI	Performance indicator for a scheduling policy in a project library
ProGen	Project generation tool used to create the PSPLIB project libraries
RAN	Random scheduling policy
RanGen	First version of the random project generation tool used to create the RG30 library
RanGen2	Project generation tool derived from its previous version RanGen and used to create the RG30 library
RCPSP	Resource constrained project scheduling problem
RF	Rescale factor for the activity durations
RG30	Project library with 30 activities per project generated with the RanGen project generator tool

SGS	Schedule generation scheme
SIO	Shortest imminent operation (scheduling policy)
TS	Tabu search

Bibliography

Abbasi, B., S. Shadrokh, and J. Arkat (2006). "Bi-objective resource-constrained project scheduling with robustness and makespan criteria". In: *Applied Mathematics and Computation* 180(1), pp. 146–152.

Abbeel, P., A. Coates, M. Quigley, and A. Ng (2007). "An Application of Reinforcement Learning to Aerobatic Helicopter Flight". In: *Advances in Neural Information Processing Systems*. Ed. by B. Schölkopf, J. Platt, and T. Hoffman. Vol. 19. (Last visited on 14/07/2021). MIT Press, pp. 1–8. URL: https://proceedings.neurips.cc/paper/2006/file/98c39996bf1543e974747a2549b3107c-Paper.pdf.

Abdolshah, M. (2014). "A review of resource-constrained project scheduling problems (RCPSP) approaches and solutions". In: *International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies* 5(4), p. 256.

Adamu, P. and O. Aromolaran (2018). "Machine learning priority rule (MLPR) for solving resource-constrained project scheduling problems". In: *International Multi-Conference of Engineers and Computer Scientists, Hong Kong*. (Last visited on 14/07/2021). URL: http://eprints.covenantuniversity.edu.ng/11059/.

Agarwal, A., S. Colak, J. Deane, and T. Rakes (2014). "The Task Scheduling Problem: A NeuroGenetic Approach". In: *Journal of Business and Economics Research* 12(4), p. 327.

Agarwal, A., S. Colak, and S. Erenguc (2015). "Metaheuristic methods". In: *Handbook on Project Management and Scheduling Vol. 1*. Springer International Publishing Switzerland, Cham, pp. 57–74.

Aggarwal, C. C. (2018). *Neural networks and deep learning*. Springer International Publishing Switzerland, Cham, pp. 978–3.

Aghdam, H. H. and E. Heravi (2017). "Guide to Convolutional Neural Networks". In: *Springer New York, NY* 10, pp. 978–3.

Almeida, J. H. L., L. A. R. Lopes, M. A. B. Silva, and J. L. M. Amaral (2018). "Convolutional Neural Networks applied in the monitoring of metallic parts". In: *International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil*. IEEE, pp. 1–8.

Artigues, C., R. Leus, and F. T. Nobibon (2013). "Robust optimization for resource-constrained project scheduling with uncertain activity durations". In: *Flexible Services and Manufacturing Journal* 25(1-2), pp. 175–205.

Ashtiani, B., R. Leus, and M. Aryanezhad (2011). "New competitive results for the stochastic resource-constrained project scheduling problem: exploring the benefits of pre-processing". In: *Journal of Scheduling* 14(2), pp. 157–171.

Ballestín, F. (2007). "When it is worthwhile to work with the stochastic RCPSP?" In: *Journal of Scheduling* 10(3), pp. 153–166.

Ballestín, F. and R. Leus (2009). "Resource-constrained project scheduling for timely project completion with stochastic activity durations". In: *Production and Operations Management* 18(4), pp. 459–474.

Benbrahim, H. and J. A. Franklin (1997). "Biped dynamic walking using reinforcement learning". In: *Robotics and Autonomous Systems* 22(3-4), pp. 283–302.

Bergstra, J. and Y. Bengio (2012). "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13(2), pp. 281–305.

Bergstra, J., D. Yamins, and D. D. Cox (2013). "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures". In: *International Conference on Machine Learning, Atlanta, USA*, pp. 115–123.

Berthaut, F., R. Pellerin, N. Perrier, and A. Hajji (2014). "Time-cost trade-offs in resource-constraint project scheduling problems with overlapping modes". In: *International Journal of Project Organisation and Management* 6(3), pp. 215–236.

Bianchi, L., M. Dorigo, L. M. Gambardella, and W. J. Gutjahr (2009). "A survey on metaheuristics for stochastic combinatorial optimization". In: *Natural Computing* 8(2), pp. 239–287.

Bishop, C. M (2006). *Pattern recognition and machine learning*. Springer New York.

Blazewicz, J., J. K. Lenstra, and A. H. G. R. Kan (1983). "Scheduling subject to resource constraints: classification and complexity". In: *Discrete Applied Mathematics* 5(1), pp. 11–24.

Bouffard, V. and J. A. Ferland (2007). “Improving simulated annealing with variable neighborhood search to solve the resource-constrained scheduling problem”. In: *Journal of Scheduling* 10(6), pp. 375–386.

Bouleimen, K. and H. Lecocq (2003). “A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version”. In: *European Journal of Operational Research* 149(2), pp. 268–281.

Brochu, E., V. M. Cora, and N. De Freitas (2010). “A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Madrid, Spain*.

Bruni, M. E., P. Beraldi, and F. Guerriero (2015). “The stochastic resource-constrained project scheduling problem”. In: *Handbook on Project Management and Scheduling Vol. 2*. Springer International Publishing Switzerland, Cham, pp. 811–835.

Carlier, J. and A. Moukrim (2015). “Storage resources”. In: *Handbook on Project Management and Scheduling Vol. 1*. Springer International Publishing Switzerland, Cham, pp. 177–189.

Cattrysse, D. G. and L. N. Van Wassenhove (1992). “A survey of algorithms for the generalized assignment problem”. In: *European Journal of Operational Research* 60(3), pp. 260–272. ISSN: 0377-2217.

Chand, S., H. K. Singh, and T. Ray (2016). “Finding robust solutions for resource constrained project scheduling problems involving uncertainties”. In: *IEEE Congress on Evolutionary Computation (CEC), Vancouver*. IEEE, pp. 225–232.

Chen, L. and Z. Zhang (2016). “Preemption resource-constrained project scheduling problems with fuzzy random duration and resource availabilities”. In: *Journal of Industrial and Production Engineering* 33(6), pp. 373–382.

Claesen, M. and B. De Moor (2015). “Hyperparameter search in machine learning”. In: *arXiv preprint arXiv:1502.02127*. (Last visited on 14/07/2021). URL: `https://arxiv.org/abs/1502.02127`.

Colling, D., J. Dziedzitz, K. Furmans, P. Hopfgarten, and K. Markert (2018). “Progress in Autonomous Picking as Demonstrated by the Amazon Robotic Challenge”. In: *Progress in Material Handling Research, 15TH IMHRC Proceedings, Savannah, Georgia, USA*.

Creemers, S. (2015). “Minimizing the expected makespan of a project with stochastic activity durations under resource constraints”. In: *Journal of Scheduling* 18(3), pp. 263–273.

Da Silva, I. N., D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves (2017). “Artificial neural network architectures and training processes”. In: *Artificial neural networks*. Springer International Publishing Switzerland, Cham, pp. 21–28.

Davis, E. W. (1973). “Project scheduling under resource constraints - historical review and categorization of procedures”. In: *AIIE Transactions* 5(4), pp. 297–313.

Demeulemeester, E. and W. Herroelen (1992). “A branch-and-bound procedure for the multiple resource-constrained project scheduling problem”. In: *Management Science* 38(12), pp. 1803–1818.

Demeulemeester, E. and W. Herroelen (2002). *Project scheduling: a research handbook*. Kluwer Academic Publishers, Dordrecht.

Demeulemeester, E., M. Vanhoucke, and W. Herroelen (2003). “RanGen: A random network generator for activity-on-the-node networks”. In: *Journal of Scheduling* 6(1), pp. 17–38.

Duan, Y., B. Cui, and H. Yang (2008). “Robot navigation based on fuzzy RL algorithm”. In: *International Symposium on Neural Networks, Moscow, Russia*. Ed. by Berlin Heidelberg Springer, pp. 391–399.

Efendigil, T., S. Önüt, and C. Kahraman (2009). “A decision support system for demand forecasting with artificial neural networks and neuro-fuzzy models: A comparative analysis”. In: *Expert Systems with Applications* 36(3), pp. 6697–6707.

Eigner, M. and R. Stelzer (2009). *Product Lifecycle Management*. Springer Science & Business Media Berlin Heidelberg.

Elmaghraby, S. E. (1977). *Activity networks: Project planning and control by network models*. John Wiley & Sons, New York.

Evans, J. R. and D. L. Olson (2001). *Introduction to Simulation and Risk Analysis*. Prentice Hall PTR, New Jersey.

Fang, C., R. Kolisch, L. Wang, and C. Mu (2015). “An estimation of distribution algorithm and new computational results for the stochastic resource-constrained project scheduling problem”. In: *Flexible Services and Manufacturing Journal* 27(4), pp. 585–605.

French, S. (1986). *Decision theory: an introduction to the mathematics of rationality*. Halsted Press, Sidney.

Goldsborough, P. (2016). "A tour of tensorflow". In: *arXiv preprint arXiv:1610.01178*. (Last visited on 14/07/2021). URL: https://arxiv.org/abs/1610.01178.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT press Cambridge, MA, USA.

Goodfellow, I., Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet (2013). "Multi-digit number recognition from street view imagery using deep convolutional neural networks". In: *arXiv preprint arXiv:1312.6082*. (Last visited on 14/07/2021). URL: https://arxiv.org/abs/1312.6082.

Habibi, F., F. Barzinpour, and S. Sadjadi (2018). "Resource-constrained project scheduling problem: review of past and recent developments". In: *Journal of Project Management* 3(2), pp. 55–88.

Hafner, R. and M. Riedmiller (2003). "Reinforcement learning on an omnidirectional mobile robot". In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), Las Vegas, NV, USA*. Vol. 1. IEEE, pp. 418–423.

Hartmann, S. (1999). *Project scheduling under limited resources: models, methods, and applications*. Vol. 478. Springer Science & Business Media New York.

Hartmann, S. and D. Briskorn (2010). "A survey of variants and extensions of the resource-constrained project scheduling problem". In: *European Journal of Operational Research* 207(1), pp. 1–14.

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition, Las Vegas, NV, USA*, pp. 770–778.

Herroelen, W. and R. Leus (2005). "Project scheduling under uncertainty: Survey and research potentials". In: *European Journal of Operational Research* 165(2), pp. 289–306.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov (2012). "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR: The Computing Research Repository*. (Last visited on 14/07/2021). URL: https://arxiv.org/abs/1207.0580.

Hofmann, C., N. Brakemeier, C. Krahe, N. Stricker, and G. Lanza (2018). "The Impact of Routing and Operation Flexibility on the Performance of Matrix Production Compared to a Production Line". In: *Congress of the German Academic Association for Production Technology, Aachen, Germany*. Ed. by Cham Springer, pp. 155–165.

Hosu, I. A. and T. Rebedea (2016). "Playing atari games with deep reinforcement learning and human checkpoint replay". In: *CoRR: The Computing Research Repository*. (Last visited on 14/07/2021). URL: https://arxiv.org/abs/1607.05077.

Hubel, D. H. and T. N. Wiesel (1959). "Receptive fields of single neurones in the cat's striate cortex". In: *The Journal of Physiology* 148(3), pp. 574–591.

Jędrzejowicz, P. and E. Ratajczak-Ropel (2014). "Reinforcement learning strategies for A-team solving the resource-constrained project scheduling problem". In: *Neurocomputing* 146, pp. 301–307.

Jędrzejowicz, P. and E. Ratajczak-Ropel (2017). "Reinforcement learning strategy for solving the MRCPSP by a team of agents". In: *International Conference on Intelligent Decision Technologies, Vilamoura, Portugal*. Springer International publishing, Cham, pp. 537–548.

Jia, Q. and Y. Seo (2013). "An improved particle swarm optimization for the resource-constrained project scheduling problem". In: *The International Journal of Advanced Manufacturing Technology* 67(9-12), pp. 2627–2638.

Johansson, R. (2019). "Introduction to computing with python". In: *Numerical Python*. Springer Science+Business Media New York, pp. 1–41.

Kerzner, H. (2017). *Project management: a systems approach to planning, scheduling, and controlling*. John Wiley & Sons, Hoboken, New Jersey, USA.

Ketkar, N. and Eder S. (2017). *Deep Learning with Python*. Vol. 1. Springer Science and Business Media New York.

Khan, S., H. Rahmani, S. A. A. Shah, and M. Bennamoun (2018). "A guide to convolutional neural networks for computer vision". In: *Synthesis Lectures on Computer Vision* 8(1), pp. 1–207.

Klein, R. (1999). *Scheduling of resource-constrained projects*. Vol. 10. Springer Science & Business Media New York: p. 2.

Kober, J., A. Bagnell, and J. Peters (2013). "Reinforcement learning in robotics: a survey". In: *The International Journal of Robotics Research* 32(11), pp. 1238–1274.

Kochak, A. and S. Sharma (2015). “Demand forecasting using neural network for supply chain management”. In: *International Journal of Mechanical Engineering and Robotics Research* 4(1), pp. 96–104.

Kolisch, R. (1996). “Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation”. In: *European Journal of Operational Research* 90(2), pp. 320–333.

Kolisch, R. and S. Hartmann (1999). “Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis”. In: *Project scheduling*. Springer Science+Business Media New York, pp. 147–178.

Kolisch, R. and S. Hartmann (2006). “Experimental investigation of heuristics for resource-constrained project scheduling: An update”. In: *European Journal of Operational Research* 174(1), pp. 23–37.

Kolisch, R. and R. Padman (2001). “An integrated survey of deterministic project scheduling”. In: *Omega* 29(3), pp. 249–272.

Kool, W., H. Van Hoof, and M. Welling (2018). “Attention, Learn to Solve Routing Problems!” In: *International Conference on Learning Representations, Vancouver*. Elsevier Amsterdam.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. Vol. 25, pp. 1097–1105.

Kyriakidis, T. S., G. M. Kopanos, and M. C. Georgiadis (2012). “MILP formulations for single-and multi-mode resource-constrained project scheduling problems”. In: *Computers and Chemical Engineering* 36, pp. 369–385.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86(11), pp. 2278–2324.

Leung, H. C. (1995). “Neural networks in supply chain management”. In: *Proceedings for Operating Research and the Management Sciences, Singapore*, pp. 347–352.

Li, H. and N. K. Womer (2015). “Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming”. In: *European Journal of Operational Research* 246(1), pp. 20–33.

Library hyperopt (2020). URL: https://github.com/hyperopt/hyperopt (visited on 07/07/2020).

Library multiprocessing (2020). URL: https://docs.python.org/3/library/multiprocessing.html (visited on 07/07/2020).

Library numpy (2020). URL: https://numpy.org/ (visited on 07/07/2020).

Lutz, M. (2013). *Learning python: Powerful object-oriented programming*. O'Reilly Media, Inc., Sebastopol, California, USA.

MacKay, D. J. C. and D. J. C. Mac Kay (2003). *Information theory, inference and learning algorithms*. Cambridge university press, Cambridge, UK.

Mahadevan, S. and G. Theocharous (1998). "Optimizing Production Manufacturing Using Reinforcement Learning." In: *FLAIRS Conference, Florida*. Vol. 372, p. 377.

Mao, H., M. Alizadeh, I. Menache, and S. Kandula (2016). "Resource management with deep reinforcement learning". In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks, Atlanta*, pp. 50–56.

McCulloch, W. S. and W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5(4), pp. 115–133.

Merkle, D., M. Middendorf, and H. Schmeck (2002). "Ant colony optimization for resource-constrained project scheduling". In: *IEEE Transactions on Evolutionary Computation* 6(4), pp. 333–346.

Mirzaei, O. and M. Akbarzadeh (2012). "A novel learning algorithm based on a multi-agent structure for solving multi-mode resource-constrained project scheduling problem". In: *Computer Science and its Applications*. Springer Science+Business Media Dordrecht, pp. 231–242.

Mitchell, T. M. (1997). *Machine learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602*. (Last visited on 14/07/2021). URL: https://arxiv.org/abs/1312.5602.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518(7540), pp. 529–533.

Moehring, R. H. (1984). “Minimizing costs of resource requirements in project networks subject to a fixed completion time”. In: *Operations Research* 32(1), pp. 89–120.

Moehring, R. H., A. S. Schulz, F. Stork, and M. Uetz (2003). “Solving project scheduling problems by minimum cut computations”. In: *Management Science* 49(3), pp. 330–350.

Munns, A. K. and B. F. Bjeirmi (1996). “The role of project management in achieving project success”. In: *International Journal of Project Management* 14, pp. 81–87.

Myszkowski, P. B., M. E. Skowronski, L. P. Olech, and K. Oslizlo (2015). “Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem”. In: *Soft Computing* 19(12), pp. 3599–3619.

Neron, E. (2002). “Lower bounds for the multi-skill project scheduling problem”. In: *Proceeding of the Eighth International Workshop on Project Management and Scheduling, Valencia, Spain*, pp. 274–277.

Neumann, K., C. Schwindt, and J. Zimmermann (2012). *Project scheduling with time windows and scarce resources: temporal and resource-constrained project scheduling with regular and nonregular objective functions*. Springer Science & Business Media New York.

Nonobe, K. and T. Ibaraki (2002). “Formulation and tabu search algorithm for the resource constrained project scheduling problem”. In: *Essays and surveys in metaheuristics*. Springer Science+Business Media New York, pp. 557–588.

Pagani, P., G. Fischer, I. Farquhar, R. Skilton, and M. Mittwollen (2019). “A logistical simulation tool to quantitatively evaluate the effect of different maintenance solutions on the total maintenance downtime for fusion reactors”. In: *Fusion Engineering and Design* 141, pp. 121–124.

Pagani, P. and F. Pfann (2020). “Deep Neural Networks for the Scheduling of Resource-Constrained Activity Sequences: A Preliminary Investigation”. In: *Logistics Journal: Proceedings, Rostock, Germany* 12.

Palacio, J. D. and O. L. Larrea (2017). “A lexicographic approach to the robust resource-constrained project scheduling problem”. In: *International Transactions in Operational Research* 24(1-2), pp. 143–157.

Patterson, J. H. (1973). “Alternate methods of project scheduling with limited resources”. In: *Naval Research Logistics Quarterly* 20(4), pp. 767–784.

Patterson, J. H. (1984). “A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem”. In: *Management Science* 30(7), pp. 854–867.

Pelikan, M., D. E. Goldberg, and E. Cantu-Paz (1999). “BOA: The Bayesian optimization algorithm”. In: *Proceedings of the genetic and evolutionary computation conference GECCO-99, Orlando, Florida, USA*. Vol. 1, pp. 525–532.

Pons, J., T. Lidy, and X. Serra (2016). “Experimenting with musically motivated convolutional neural networks”. In: *14th international workshop on content-based multimedia indexing (CBMI), Bucharest, Romania*. IEEE, pp. 1–6.

Pontrandolfo, P., A. Gosavi, O. G. Okogbaa, and T. K. Das (2002). “Global supply chain management: a reinforcement learning approach”. In: *International Journal of Production Research* 40(6), pp. 1299–1317.

Radermacher, F. J. (1985). “Scheduling of project networks”. In: *Annals of Operations Research* 4(1), pp. 227–252.

Random Network Generation Website (2020). URL: `https://www.projectmanagement.ugent.be/research/data/RanGen` (visited on 05/18/2020).

Repository of the P-30-120-10 project generator and test projects (2021). URL: `https://www.ifl.kit.edu/repository_project_generator.php` (visited on 05/01/2021).

Riedmiller, M., T. Gabel, R. Hafner, and S. Lange (2009). “Reinforcement learning for robot soccer”. In: *Autonomous Robots* 27(1), pp. 55–73.

Rose, K. H. (2013). “A Guide to the Project Management Body of Knowledge (PMBOK Guide) Fifth Edition”. In: *Project Management Journal* 44(3), e1–e1.

Rosenhead, J., M. Elton, and S. K. Gupta (1972). “Robustness and optimality as criteria for strategic decisions”. In: *Journal of the Operational Research Society* 23(4), pp. 413–431.

Rostami, S., S. Creemers, and R. Leus (2015). “New benchmark results for the stochastic resource-constrained project scheduling problem”. In: *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Singapore*. IEEE, pp. 204–208.

Rostami, S., S. Creemers, and R. Leus (2018). "New strategies for stochastic resource-constrained project scheduling". In: *Journal of Scheduling* 21(3), pp. 349–365.

Ruder, S. (2016). "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747*. (Last visited on 14/07/2021). URL: https://arxiv.org/abs/1609.04747.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323(6088), pp. 533–536.

Sabzehparvar, M. and S. M. Seyed-Hosseini (2008). "A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags". In: *The Journal of Supercomputing* 44(3), pp. 257–273.

Schwindt, C. and J. Zimmermann (2015a). *Handbook on project management and scheduling vol. 1*. Springer International Publishing Switzerland, Cham: pp. 2-17.

Schwindt, C. and J. Zimmermann (2015b). *Handbook on project management and scheduling vol. 2*. Springer International Publishing Switzerland, Cham.

Scott, E. T. and Srivatsan P. (2006). "Moore's law: the future of Si microelectronics". In: *Materials Today* 9(6), pp. 20–25.

Silva, N., L. M. D. F. Ferreira, C. Silva, V. Magalhães, and P. Neto (2017). "Improving supply chain visibility with artificial neural networks". In: *Procedia Manufacturing* 11, pp. 2083–2090.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and M. Lanctot (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529(7587), p. 484.

Simonyan, K. and A. Zisserman (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, Conference Track Proceedings*.

Slowinski, R. (1980). "Two approaches to problems of resource allocation among project activities - a comparative study". In: *Journal of the Operational Research Society* 31(8), pp. 711–723.

Snoek, J., H. Larochelle, and R. P. Adams (2012). "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems* 25, pp. 2951–2959.

Sprecher, A. and R. Kolisch (1996). "PSPLIB a project scheduling problem library". In: *European Journal of Operational Research* 96, pp. 205–216.

Stork, F. (2001). "Stochastic resource-constrained project scheduling". (Last visited on 14/07/2021). Doctoral Thesis. Berlin: Technical University Berlin, Faculty II - Mathematics and Science. DOI: 10.14279/depositonce-398. URL: http://dx.doi.org/10.14279/depositonce-398.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT press, Cambridge, MA, USA.

Ta, X. and Y. Wei (2018). "Research on a dissolved oxygen prediction method for recirculating aquaculture systems based on a convolution neural network". In: *Computers and Electronics in Agriculture* 145, pp. 302–310.

Thomas, P. R. and S. Salhi (1998). "A tabu search approach for the resource constrained project scheduling problem". In: *Journal of Heuristics* 4(2), pp. 123–139.

Thrun, S. (1995). "An approach to learning mobile robot navigation". In: *Robotics and Autonomous systems* 15(4), pp. 301–319.

Valls, V., F. Ballestin, and S. Quintanilla (2008). "A hybrid genetic algorithm for the resource-constrained project scheduling problem". In: *European Journal of Operational Research* 185(2), pp. 495–508.

Van de Vonder, S., E. Demeulemeester, W. Herroelen, and R. Leus (2005). "The use of buffers in project management: The trade-off between stability and makespan". In: *International Journal of Production Economics* 97(2), pp. 227–240.

Van den Berg, J. P. (1999). "A literature survey on planning and control of warehousing systems". In: *IIE Transactions* 31(8), pp. 751–762.

Van Peteghem, V. and M. Vanhoucke (2011). "Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem". In: *Journal of Heuristics* 17(6), pp. 705–728.

Vanhoucke, M. (2012). *Project management with dynamic scheduling*. Springer Berlin Heidelberg: pp. 1-14.

Vanhoucke, M. (2016). *Integrated project management sourcebook*. Springer International Publishing Switzerland, Cham.

Vanhoucke, M., J. Coelho, D. Debels, B. Maenhout, and L. V. Tavares (2008). "An evaluation of the adequacy of project network generators with systematically sampled networks". In: *European Journal of Operational Research* 187(2), pp. 511–524.

Walsh, J., N. O' Mahony, S. Campbell, A. Carvalho, L. Krpalkova, G. Velasco-Hernandez, S. Harapanahalli, and D. Riordan (2019). "Deep learning vs. traditional computer vision". In: *Proceedings of the computer vision conference (CVC 2019), Las Vegas, USA*, pp. 25–26.

Wang, K., J. Chi, and E. Wan (1993). "Decision making of project under fuzzy information". In: *Journal of the Chinese Institute of Engineers* 16(4), pp. 533–541.

Wauters, T., K. Verbeeck, G. V. Berghe, and P. De Causmaecker (2011). "Learning agents for the multi-mode project scheduling problem". In: *Journal of the Operational Research Society* 62(2), pp. 281–290.

Wauters, T., K. Verbeeck, P. De Causmaecker, and G. V. Berghe (2015). "A learning-based optimization approach to multi-project scheduling". In: *Journal of Scheduling* 18(1), pp. 61–74.

Wiesemann, W., D. Kuhn, and B. Rustem (2010). "Maximizing the net present value of a project under uncertainty". In: *European Journal of Operational Research* 202(2), pp. 356–367.

Zamani, R. (2013). "A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem". In: *European Journal of Operational Research* 229(2), pp. 552–559.

Zhang, H., H. Xu, and W. Peng (2008). "A genetic algorithm for solving RCPSP". In: *International Symposium on Computer Science and Computational Technology, Shanghai*. Vol. 2. IEEE, pp. 246–249.

Zhang, W. and T. G. Dietterich (1995). "A reinforcement learning approach to job-shop scheduling". In: *International Joint Conferences on Artificial Intelligence, Montreal, Canada*. Vol. 95. Citeseer, pp. 1114–1120.

Zheng, H., L. Wang, and X. Zheng (2017). "Teaching-learning-based optimization algorithm for multi-skill resource constrained project scheduling problem". In: *Soft Computing* 21(6), pp. 1537–1548.

List of Figures

List of Tables

A Intermediate numerical results of the hyperparameter tuning

This appendix presents for each of the six neural network structures all the 50 tested hyperparameter combinations chosen by the Bayesian optimization algorithm along with the correspondent AIP. In the Tables from A.1 to A.6, each row represents a tested hyperparameter combination and the row order corresponds to the one followed during the Bayesian optimization. The bolded row highlights the best combination during each hyperparameter tuning.

Table A.1: Intermediate results of the hyperparameter tuning for the NN-FC neural network structure

Comb.	R_{max}	$NoL_{NN_1+NN_{final}}$	$NoN_{NN_1+NN_{final}}$	N_{epochs}	α	σ	$AIP[\%]$
1	6	9	1024	17	$10^{-4.4}$	.75	1.185
2	7	11	512	24	$10^{-4.5}$	.56	1.325
3	7	10	512	22	$10^{-4.5}$	.85	0.821
4	7	12	256	10	$10^{-3.8}$	.51	0.508
5	9	11	256	15	$10^{-4.2}$	.81	0.943
6	8	10	256	17	$10^{-3.4}$	.78	0.302
7	9	12	1024	14	$10^{-4.5}$	.65	1.126
8	7	11	512	23	$10^{-4.5}$	.84	0.942
9	8	8	1024	21	$10^{-4.7}$	.88	1.313
10	8	10	256	23	$10^{-3.3}$	.58	0.302
11	7	10	1024	11	$10^{-4.2}$	.6	1.387
12	7	9	512	18	10^{-5}	.7	0.330
13	7	12	1024	22	$10^{-3.8}$	.84	0.907
14	6	8	1024	23	$10^{-4.9}$	.62	0.763
15	8	9	1024	17	$10^{-4.6}$	.89	0.958
16	9	11	512	22	$10^{-4.5}$	.76	0.788
17	6	11	1024	19	10^{-4}	.6	1.500
18	6	11	512	20	$10^{-3.6}$	.5	0.708
19	6	8	1024	21	$10^{-4.8}$	.51	0.296
20	7	11	512	14	$10^{-3.7}$	.83	1.179
21	6	10	1024	21	10^{-4}	.65	1.091
22	9	11	256	11	$10^{-4.3}$	.67	0.332
23	6	11	512	10	$10^{-3.5}$	.61	0.760
24	6	9	1024	19	10^{-4}	.66	1.408
25	6	9	1024	13	$10^{-4.1}$	.67	1.525

Comb.	R_{max}	$NoL_{NN_1+NN_{final}}$	$NoN_{NN_1+NN_{final}}$	N_{epochs}	α	σ	AIP[%]
26	7	10	1024	12	$10^{-4.3}$	.64	1.543
27	6	9	1024	19	10^{-4}	.57	1.398
28	8	11	1024	16	$10^{-3.1}$	.73	0.256
29	7	12	1024	12	$10^{-3.9}$	.54	1.219
30	6	10	1024	13	$10^{-3.8}$	.52	1.262
31	6	10	1024	14	$10^{-3.7}$	.63	1.459
32	7	10	256	10	$10^{-4.1}$	.59	0.866
33	**7**	**9**	**512**	**14**	$10^{-4.5}$	**.71**	**1.551**
34	7	8	1024	10	$10^{-4.1}$	.72	1.521
35	8	11	1024	12	$10^{-4.3}$	.68	0.085
36	6	8	1024	16	$10^{-4.7}$	.75	1.119
37	6	10	256	13	$10^{-3.9}$	.7	1.401
38	7	9	1024	16	$10^{-3.5}$	.69	1.093
39	8	12	256	14	$10^{-4.1}$	.54	0.701
40	9	11	1024	14	$10^{-3.7}$	.57	0.634
41	7	11	512	11	$10^{-4.6}$	.81	0.998
42	7	10	512	12	$10^{-4.4}$	.74	0.664
43	8	9	1024	15	$10^{-4.4}$	.77	0.715
44	7	10	1024	18	$10^{-3.8}$	.79	1.140
45	6	12	1024	11	$10^{-4.2}$	.65	1.466
46	7	11	512	14	$10^{-3.2}$	.86	0.727
47	9	12	256	12	$10^{-3.9}$	.56	0.784
48	7	10	512	13	10^{-5}	.73	0.889
49	6	8	1024	10	$10^{-3.4}$	.67	0.223
50	8	11	1024	17	$10^{-4.6}$	.7	0.591

Table A.2: Intermediate results of the hyperparameter tuning for the NN-FC-FRU neural network structure

Comb.	R_{max}	T	NoL_{NN_1}	NoN_{NN_1}	NoL_{NN_2}	NoF_{NN_2}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
1	8	7	5	1024	5	64	6	1024	13	10^{-4}	.64	1.601
2	9	15	5	512	6	32	5	256	24	$10^{-3.3}$	.85	0.863
3	7	12	6	1024	5	64	4	1024	17	$10^{-3.1}$	.77	0.201
4	7	12	6	512	6	64	6	256	13	10^{-5}	.7	0.631
5	9	13	4	512	6	128	5	1024	22	10^{-4}	.68	1.891
6	9	9	4	256	5	64	6	256	19	$10^{-3.5}$	.9	1.664
7	8	13	6	1024	4	128	6	1024	23	$10^{-3.2}$	.73	0.240
8	9	10	6	1024	5	32	6	512	22	$10^{-3.9}$	.59	1.165
9	8	9	5	512	5	128	6	1024	11	$10^{-3.9}$	.74	1.364
10	7	13	5	256	4	128	5	1024	21	$10^{-4.2}$	.77	1.130
11	6	12	4	256	5	32	5	512	22	10^{-4}	.5	0.909
12	7	9	5	256	6	32	5	256	22	$10^{-3.6}$	.74	1.136
13	7	6	5	512	6	128	5	512	12	$10^{-4.1}$	.8	1.908
14	7	6	4	1024	4	128	6	1024	15	$10^{-3.7}$	.55	0.300
15	8	8	5	1024	4	64	4	512	16	$10^{-3.6}$	.55	0.013
16	6	5	4	1024	4	32	5	1024	19	$10^{-3.3}$	.8	0.380
17	8	10	4	256	4	128	5	512	14	$10^{-3.9}$	.58	0.583
18	8	7	4	256	6	32	5	256	13	$10^{-3.9}$	.83	1.460
19	7	10	5	256	5	64	5	1024	21	$10^{-4.8}$	.87	1.245
20	9	6	5	1024	5	32	6	1024	19	$10^{-4.6}$	.71	1.823
21	6	15	4	512	6	128	4	512	25	$10^{-4.3}$	.65	1.522
22	6	14	5	512	6	128	4	512	11	$10^{-4.4}$	.66	1.308
23	7	11	6	512	6	128	5	512	10	$10^{-4.2}$	.8	1.661
24	6	14	5	512	6	128	4	512	18	$10^{-4.6}$	.63	1.461
25	7	5	4	512	6	128	5	512	25	$10^{-4.5}$	.68	1.801

Comb.	R_{max}	T	NoL_{NN_1}	NoN_{NN_1}	NoL_{NN_2}	NoF_{NN_2}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
26	9	11	5	512	6	128	5	512	20	$10^{-4.2}$	.78	1.768
27	8	14	6	512	6	128	4	1024	17	$10^{-4.8}$	.88	1.479
28	6	8	4	512	6	128	5	1024	10	$10^{-3.7}$	.82	2.250
29	6	7	5	512	6	128	5	512	10	$10^{-3.7}$	.83	1.211
30	6	8	5	512	6	128	4	1024	12	$10^{-3.4}$	.9	1.481
31	6	6	5	512	6	128	5	256	11	$10^{-3.7}$	.84	1.764
32	7	7	6	512	5	128	4	1024	15	$10^{-4.1}$	.81	2.137
33	6	8	6	512	5	64	4	1024	15	10^{-3}	.86	0.869
34	7	7	6	512	5	128	4	1024	14	$10^{-3.8}$	.82	1.792
35	7	8	6	512	5	64	4	1024	16	$10^{-3.5}$	.76	1.143
36	6	9	6	512	4	128	4	1024	14	$10^{-3.1}$	.9	0.309
37	6	5	6	512	5	64	6	1024	12	$10^{-4.1}$	.72	1.579
38	7	7	4	512	4	128	5	256	16	$10^{-3.3}$	.85	1.298
39	8	11	6	512	5	64	6	1024	18	$10^{-4.4}$	.88	1.925
40	7	9	4	1024	5	128	4	1024	15	10^{-5}	.75	2.263
41	6	9	4	1024	4	128	6	1024	10	10^{-5}	.69	0.684
42	7	12	4	1024	5	32	5	256	13	$10^{-4.7}$	.75	0.482
43	8	10	4	1024	5	32	5	1024	24	10^{-5}	.63	1.239
44	6	11	4	1024	4	32	6	1024	17	$10^{-3.4}$	.78	0.597
45	7	9	4	1024	5	32	5	1024	13	$10^{-3.1}$	.73	1.970
46	9	8	4	1024	6	32	4	256	20	$10^{-4.9}$	.67	0.621
47	8	10	4	1024	4	32	5	1024	11	$10^{-3.8}$	.6	1.399
48	7	9	4	256	4	32	4	1024	12	$10^{-3.5}$	.78	1.172
49	**6**	**12**	**4**	**1024**	**6**	**64**	**6**	**256**	**15**	10^{-4}	**.7**	**2.478**
50	7	12	4	1024	5	64	6	256	23	10^{-4}	.7	2.163

Table A.3: Intermediate results of the hyperparameter tuning for the NN-CONV1D neural network structure

Comb.	R_{max}	NoL_{NN_1}	NoF_{NN_1}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
1	9	6	32	5	256	15	$10^{-4.2}$	.79	1.700
2	7	5	32	6	512	11	$10^{-3.8}$	.55	1.070
3	6	4	128	6	1024	17	$10^{-4.4}$	.75	2.089
4	8	5	64	5	512	15	$10^{-3.7}$	.78	2.206
5	6	5	128	4	512	23	$10^{-4.5}$	.55	2.371
6	8	4	32	6	1024	17	$10^{-3.4}$	.82	0.843
7	9	4	128	5	512	14	$10^{-4.5}$	.65	2.084
8	7	5	64	6	1024	25	$10^{-4.5}$	.85	1.559
9	7	4	64	6	512	18	10^{-5}	.7	0.859
10	8	6	64	5	1024	23	$10^{-4.5}$	.84	1.802
11	8	5	32	4	1024	22	$10^{-3.3}$	.58	0.467
12	7	5	128	6	1024	20	$10^{-4.8}$	.88	2.164
13	7	6	128	5	512	23	$10^{-3.8}$	.84	1.762
14	6	5	128	4	512	19	10^{-4}	.66	2.447
15	8	4	32	5	512	17	$10^{-4.6}$	.89	1.887
16	9	5	64	5	1024	22	$10^{-4.5}$	.76	1.528
17	6	5	64	6	1024	20	$10^{-3.6}$	.5	1.435
18	6	4	128	4	256	24	$10^{-4.9}$	.62	1.492
19	7	5	128	4	512	18	$10^{-4.1}$	.57	2.338
20	7	4	32	4	256	25	$10^{-4.8}$	.51	0.852
21	6	6	32	4	256	13	$10^{-3.8}$	.52	2.282
22	6	5	128	4	256	26	10^{-4}	.65	1.922
23	6	5	128	6	512	18	10^{-4}	.6	2.596
24	6	5	128	6	512	13	$10^{-4.1}$	.63	2.641
25	**7**	**6**	**128**	**4**	**512**	**11**	$10^{-3.9}$	**.67**	**2.697**

Comb.	R_{max}	NoL_{NN_1}	NoF_{NN_1}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
26	6	6	64	4	512	12	$10^{-3.5}$	.61	1.602
27	7	5	64	4	512	10	$10^{-4.2}$	.6	2.402
28	9	6	128	4	256	11	$10^{-3.7}$	.57	1.359
29	8	5	128	4	512	16	$10^{-3.2}$	.73	0.078
30	8	4	128	4	512	12	$10^{-4.3}$	.64	2.667
31	7	6	128	4	512	13	$10^{-3.9}$	.54	2.192
32	9	5	32	5	256	12	$10^{-4.3}$	.67	0.981
33	8	6	128	5	512	11	$10^{-3.3}$	.68	0.471
34	6	4	32	4	512	13	$10^{-3.9}$	.63	2.403
35	7	5	64	5	512	10	$10^{-3.9}$	.65	2.718
36	9	6	32	6	512	14	$10^{-4.1}$	.54	1.464
37	7	5	64	5	256	13	$10^{-4.4}$	.72	1.303
38	7	5	32	5	512	10	$10^{-4.1}$	.59	1.689
39	6	4	128	4	512	16	$10^{-3.5}$	.69	2.059
40	6	6	128	6	512	17	$10^{-4.7}$	.75	1.959
41	6	5	128	5	1024	14	$10^{-3.7}$	.63	2.462
42	7	4	64	4	1024	18	$10^{-3.8}$	.79	1.974
43	7	5	64	4	1024	14	$10^{-3.2}$	.86	1.507
44	8	6	64	4	512	10	$10^{-4.6}$	.81	1.803
45	8	6	128	6	256	17	$10^{-4.6}$	.7	0.360
46	8	4	128	6	256	15	$10^{-4.4}$	.77	1.485
47	6	5	128	5	512	10	$10^{-3.4}$	.67	0.719
48	9	6	32	5	512	12	$10^{-3.9}$	.56	1.569
49	7	5	64	4	1024	13	10^{-5}	.73	1.674
50	6	5	128	4	512	12	$10^{-4.2}$	.65	2.572

Table A.4: Intermediate results of the hyperparameter tuning for the NN-CONV1D-FRU neural network structure

Comb.	R_{max}	T	NoL_{NN_1}	NoF_{NN_1}	NoL_{NN_2}	NoF_{NN_2}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
1	7	6	6	32	5	128	6	512	10	$10^{-3.8}$	.55	2.397
2	7	13	5	128	5	64	4	512	24	$10^{-4.5}$	.55	3.717
3	7	8	5	64	6	64	6	1024	24	$10^{-4.5}$	.85	2.908
4	8	11	5	32	5	64	6	1024	17	$10^{-3.4}$	.82	2.231
5	9	9	5	128	5	128	5	512	14	$10^{-4.5}$	.65	3.419
6	9	10	6	32	6	64	5	256	15	$10^{-4.2}$	.79	3.073
7	7	5	6	64	5	32	5	1024	23	$10^{-4.5}$	.84	3.124
8	8	10	5	32	4	64	4	1024	23	$10^{-3.3}$	.58	1.801
9	6	13	4	128	5	32	6	1024	17	$10^{-4.4}$	.75	3.486
10	7	10	4	64	5	128	6	512	18	10^{-5}	.7	2.232
11	7	10	4	128	6	64	6	1024	20	$10^{-4.7}$	.88	3.536
12	7	7	6	128	5	64	5	512	22	$10^{-3.8}$	.84	3.089
13	8	12	5	64	5	64	5	512	15	$10^{-3.7}$	.78	3.512
14	7	12	4	32	4	32	5	512	17	$10^{-4.6}$	.89	3.214
15	9	7	6	64	5	64	5	1024	22	$10^{-4.5}$	.76	2.872
16	6	6	5	64	5	64	4	1024	20	$10^{-3.6}$	.5	2.807
17	6	15	4	128	6	64	4	256	24	$10^{-4.9}$	.62	2.796
18	6	14	4	128	4	64	4	256	25	$10^{-4.8}$	.51	2.153
19	6	14	5	128	6	64	4	512	19	10^{-4}	.66	3.815
20	6	14	5	128	4	64	4	512	25	10^{-4}	.65	3.293
21	6	14	5	128	6	64	4	512	19	10^{-4}	.57	3.737
22	6	14	5	128	6	64	4	512	18	10^{-4}	.6	3.910
23	6	15	5	128	6	32	4	512	13	$10^{-4.1}$	.63	4.037
24	6	15	5	128	6	32	4	512	11	$10^{-3.5}$	.61	2.908
25	6	12	5	128	6	32	4	512	13	$10^{-3.8}$	.52	3.582

Comb.	R_{max}	T	NoL_{NN_1}	NoF_{NN_1}	NoL_{NN_2}	NoF_{NN_2}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
26	7	15	5	128	6	32	4	512	10	$10^{-4.2}$	.6	3.796
27	8	13	5	128	6	32	4	512	16	$10^{-3.1}$	.73	1.387
28	7	11	6	128	6	32	4	512	13	$10^{-3.9}$	.54	3.587
29	**6**	**14**	**5**	**128**	**6**	**128**	**4**	**512**	**11**	$10^{-4.1}$	**.69**	**4.049**
30	9	9	5	32	6	128	4	256	11	$10^{-4.3}$	.67	2.285
31	8	15	6	128	6	128	4	512	11	$10^{-3.3}$	.68	1.811
32	7	13	4	128	6	128	4	512	10	$10^{-4.1}$	.72	4.033
33	6	9	5	32	6	128	4	512	13	$10^{-3.9}$	.63	3.714
34	9	12	6	128	6	128	4	256	14	$10^{-3.7}$	.57	2.680
35	6	11	5	128	6	128	4	512	12	$10^{-4.3}$	.64	3.978
36	8	14	6	32	4	32	6	512	14	$10^{-4.1}$	.54	2.809
37	7	15	5	64	6	128	5	256	12	$10^{-4.4}$	.72	2.698
38	7	13	4	128	6	32	4	512	16	$10^{-3.5}$	.69	3.403
39	6	8	4	128	5	128	6	512	16	$10^{-4.7}$	.75	3.341
40	7	11	5	32	6	32	5	512	10	$10^{-4.1}$	.59	3.053
41	7	5	6	64	4	128	4	512	11	$10^{-4.6}$	.81	3.137
42	6	12	5	128	5	32	5	1024	14	$10^{-3.7}$	.63	3.814
43	8	8	4	128	6	128	6	256	15	$10^{-4.4}$	.77	2.831
44	9	13	6	32	5	32	5	512	12	$10^{-3.9}$	.56	2.923
45	7	15	5	64	6	128	4	1024	13	10^{-5}	.73	3.013
46	6	14	4	128	5	32	5	512	10	$10^{-3.4}$	.67	2.029
47	8	6	6	128	6	128	6	256	17	$10^{-4.6}$	.7	1.733
48	7	10	5	64	5	32	4	1024	14	$10^{-3.2}$	.86	2.830
49	6	13	5	128	6	32	4	512	11	$10^{-4.2}$	.65	3.955
50	7	9	5	128	5	128	4	1024	18	$10^{-3.8}$	.79	3.352

Table A.5: Intermediate results of the hyperparameter tuning for the NN-CONV2D neural network structure

Comb.	R_{max}	NoL_{NN_1}	NoF_{NN_1}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
1	7	5	64	6	1024	25	$10^{-4.5}$	.85	1.360
2	7	6	128	5	512	23	$10^{-3.8}$	.84	1.502
3	7	4	64	6	512	17	10^{-5}	.7	0.609
4	9	5	32	4	1024	22	$10^{-3.3}$	.58	0.251
5	7	5	128	6	1024	20	$10^{-4.8}$	.88	1.924
6	6	5	64	6	1024	20	$10^{-3.6}$	.5	1.175
7	6	5	128	4	512	19	10^{-4}	.66	2.269
8	8	5	64	5	1024	22	$10^{-4.5}$	.76	1.281
9	7	4	32	4	256	21	$10^{-4.8}$	.51	0.619
10	6	4	128	4	256	24	$10^{-4.9}$	.62	1.229
11	6	6	32	4	256	13	$10^{-3.8}$	.52	2.062
12	7	5	128	4	512	18	$10^{-4.1}$	.57	2.147
13	9	6	128	4	256	10	$10^{-3.7}$	.57	1.144
14	6	5	128	4	256	26	10^{-4}	.65	1.700
15	7	6	128	4	512	11	$10^{-3.9}$	.67	2.430
16	6	5	128	6	512	18	10^{-4}	.6	2.330
17	6	5	128	6	512	13	$10^{-4.1}$	.63	2.392
18	7	4	128	4	512	12	$10^{-4.3}$	.64	2.414
19	6	6	64	4	512	13	$10^{-3.5}$	.61	1.341
20	7	5	64	4	512	10	$10^{-4.2}$	.6	2.175
21	8	5	128	4	512	16	$10^{-3.2}$	.73	0.185
22	6	4	32	4	512	13	$10^{-3.9}$	.63	2.187
23	7	6	128	4	512	14	$10^{-3.9}$	.54	1.951
24	9	5	32	5	256	12	$10^{-4.3}$	.67	0.784
25	8	6	128	5	512	11	$10^{-3.3}$	.68	0.420

Comb.	R_{max}	NoL_{NN_1}	NoF_{NN_1}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
26	8	6	32	6	512	14	$10^{-4.1}$	.54	1.291
27	7	5	64	5	256	12	$10^{-4.4}$	.72	1.047
28	6	6	128	6	512	17	$10^{-4.7}$	.75	1.716
29	7	5	32	5	512	10	$10^{-4.1}$	.59	1.488
30	**6**	**4**	**128**	**5**	**512**	**10**	$10^{-3.8}$	**.74**	**2.450**
31	6	4	64	4	1024	18	$10^{-3.8}$	.79	1.773
32	7	4	128	4	512	17	$10^{-3.5}$	.69	1.793
33	8	6	64	4	512	10	$10^{-4.6}$	.81	1.604
34	9	6	32	5	512	12	$10^{-3.9}$	.56	1.353
35	6	5	128	5	1024	14	$10^{-3.7}$	.63	2.264
36	6	5	128	5	512	10	$10^{-3.4}$	.67	0.954
37	7	5	64	4	1024	14	$10^{-3.2}$	.86	1.290
38	8	6	128	6	256	17	$10^{-4.6}$	.7	0.506
39	6	5	128	4	512	12	$10^{-4.2}$	.65	2.342
40	8	4	128	6	256	15	$10^{-4.4}$	.77	1.223
41	7	5	64	4	1024	13	10^{-5}	.73	1.444
42	7	5	128	5	1024	17	$10^{-3.9}$	.77	1.873
43	7	5	64	5	512	15	$10^{-3.3}$	.85	1.390
44	8	6	64	4	512	10	$10^{-4.4}$	.82	1.654
45	8	6	128	6	256	17	$10^{-4.5}$	.71	0.606
46	8	4	128	6	256	15	$10^{-4.5}$	.75	1.273
47	6	5	128	5	512	10	$10^{-3.7}$	.65	0.978
48	9	6	32	5	512	12	$10^{-3.8}$	.54	1.453
49	7	5	64	4	1024	14	$10^{-4.9}$	.74	1.454
50	6	5	128	4	512	13	$10^{-4.3}$	.66	2.352

Table A.6: Intermediate results of the hyperparameter tuning for the NN-CONV2D-FRU neural network structure

Comb.	R_{max}	T	NoL_{NN_1}	NoF_{NN_1}	NoL_{NN_2}	NoF_{NN_2}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
1	7	6	6	32	5	128	6	512	11	$10^{-3.9}$	.55	2.292
2	7	5	6	64	5	32	5	1024	22	$10^{-4.5}$	.84	2.987
3	7	6	6	64	5	32	5	1024	19	$10^{-4.5}$	.84	2.946
4	7	8	5	64	6	64	6	1024	23	$10^{-4.4}$	.87	2.780
5	9	9	5	128	5	128	5	512	13	$10^{-4.5}$	.65	3.269
6	8	11	5	32	5	64	6	1024	16	$10^{-3.4}$	.82	2.133
7	9	10	6	32	6	64	5	256	16	$10^{-4.3}$	.79	2.938
8	7	10	4	64	5	128	6	512	18	10^{-5}	.7	2.134
9	8	11	5	32	4	64	4	1024	22	$10^{-3.3}$	.58	1.722
10	6	13	4	128	5	32	6	1024	15	$10^{-4.4}$	.76	3.333
11	8	12	5	64	5	64	5	512	16	$10^{-3.7}$	.78	3.358
12	7	10	4	128	6	64	6	1024	19	$10^{-4.8}$	.88	3.380
13	7	7	6	128	5	64	5	512	23	$10^{-3.8}$	.84	2.954
14	6	7	5	64	5	64	4	1024	21	$10^{-3.6}$	.5	2.684
15	7	11	4	32	4	32	5	512	18	$10^{-4.6}$	.89	3.073
16	9	7	6	64	5	64	5	256	23	$10^{-4.5}$	.76	2.746
17	6	14	5	128	4	64	6	256	25	$10^{-4.1}$	.65	3.148
18	6	13	4	128	6	64	6	512	23	$10^{-4.9}$	.62	2.674
19	6	14	4	128	4	64	4	512	24	$10^{-4.8}$	.51	2.058
20	6	13	5	128	6	64	4	512	18	10^{-4}	.66	3.648
21	6	12	5	128	6	32	4	512	12	$10^{-3.7}$	.52	3.425
22	6	15	5	128	6	64	4	512	17	10^{-4}	.57	3.573
23	6	15	5	128	6	32	4	512	12	$10^{-3.5}$	.61	2.781
24	7	11	6	128	6	32	4	512	14	$10^{-3.9}$	.54	3.429
25	6	14	5	128	6	64	4	512	19	10^{-4}	.61	3.738

Comb.	R_{max}	T	NoL_{NN_1}	NoF_{NN_1}	NoL_{NN_2}	NoF_{NN_2}	$NoL_{NN_{final}}$	$NoN_{NN_{final}}$	N_{epochs}	α	σ	AIP[%]
26	6	15	5	128	6	32	4	512	12	$10^{-4.2}$	.63	3.860
27	7	15	5	128	6	32	5	512	11	$10^{-4.2}$	.6	3.630
28	8	13	5	128	6	32	4	512	15	$10^{-3.1}$	.73	1.326
29	6	8	4	128	5	128	6	512	16	$10^{-4.7}$	.75	3.194
30	7	11	5	64	5	32	4	1024	14	$10^{-3.2}$	.87	2.706
31	8	15	6	128	6	128	5	512	11	$10^{-3.1}$	.68	1.732
32	6	11	5	128	5	32	5	1024	14	$10^{-3.7}$	.63	3.647
33	7	5	6	64	4	128	4	512	12	$10^{-4.6}$	.81	3.000
34	8	10	5	32	6	128	4	256	10	$10^{-4.3}$	.68	2.185
35	**7**	**12**	**5**	**64**	**5**	**128**	**5**	**512**	**12**	$10^{-4.2}$	**.67**	**3.871**
36	7	15	5	64	6	128	5	256	12	$10^{-4.4}$	.72	2.579
37	7	13	4	128	6	128	4	512	10	$10^{-4.1}$	.72	3.856
38	6	9	5	32	6	128	4	512	13	$10^{-3.8}$	.63	3.551
39	8	14	6	32	4	32	6	512	14	$10^{-4.1}$	.54	2.686
40	6	14	4	128	5	32	5	512	10	$10^{-3.4}$	.68	1.940
41	9	12	6	128	6	128	4	256	14	$10^{-3.7}$	.57	2.563
42	6	11	5	128	6	128	4	512	12	$10^{-4.3}$	.64	3.803
43	7	13	4	128	6	32	4	512	16	$10^{-3.5}$	.69	3.254
44	7	9	5	128	5	128	4	1024	18	$10^{-3.8}$	.79	3.205
45	7	11	5	32	6	32	5	512	10	$10^{-4.1}$	.59	2.919
46	8	8	4	128	6	128	6	256	15	$10^{-4.4}$	.77	2.707
47	9	13	6	32	5	32	5	512	12	$10^{-3.8}$	.57	2.795
48	7	15	5	64	6	128	4	1024	13	10^{-5}	.73	2.881
49	8	6	6	128	6	128	6	256	17	$10^{-4.6}$	.7	1.657
50	6	13	5	128	6	32	4	512	11	$10^{-4.2}$	.65	3.781