Carl von Ossietzky
Universität
Oldenburg

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Masterstudiengang Informatik

Masterarbeit

Explainability of power grid attack strategies learned by Deep Reinforcement Learning Agents

vorgelegt von

B.Sc. Torben Logemann

Gutachter:

Dr.-Ing. Eric MSP Veith
Prof. Dr. rer. nat. Sebastian Lehnhoff

Oldenburg, 31. Juli 2023

Content

1 Abstract

Learning systems have achieved remarkable success. Agents trained using Deep RL (DRL) methods, e.g., promise true resilience. However, no guarantees can yet be given for the black-box models that have been learned. For operators of Critical National Infrastructures (CNIs), this is a necessity, as no responsibility for an unknown and unverifiable control system can be assumed. Intrinsically safe learning algorithms and approximate, post-hoc interpretable models exist, but they lack either learning power or explainability. To optimize this trade-off, this thesis presents the NN2EQCDT algorithm, which directly converts a (whole) policy-based Feed-Forward DNN (FF-DNN) into a compressed Decision Tree (DT). Compression is achieved by dynamically checking the satisfiability of the paths of the transformed DTs during construction and disregarding unneeded rules. It can be further increased by using additional methods and considering invariants. It has been observed that the NN2EQCDT algorithm can drastically compress a small policy model, making it possible to exactly track the action regions to their observation regions in a plotted DT and further visualizations. The NN2EQCDT algorithm was then further evaluated by explaining a learned attacker policy model to show that one can ensure that a policy-based FF-DNN has not learned unknown unknown strategies.

2 Introduction

2.1 Motivation

2.1.1 European Climate Goals 2030

The atmosphere contains a certain amount of greenhouse gases, such as carbon dioxide. They naturally absorb some of the solar radiation reflected from the earth. Some greenhouse gases, such as carbon dioxide, methane, nitrous oxide, and fluorinated gases, are enhanced by human activities. They are emitted in widely varying amounts and have different effects on suppressing reflected radiation.

When the amount of greenhouse gases is increased, e.g., by burning fossil fuels, deforestation, or livestock production, back radiation to space is suppressed, leading to global warming. This effect is called the greenhouse effect [Eur23c].

European climate targets aim to reduce gas emissions by 55 % by 2030 to keep the increase in global average temperature below 1.5 °C compared to 1990, in line with the Paris Agreement. If the global average temperature does not rise, the consequences for the climate are comparatively small. Otherwise, this would have catastrophic effects on the climate and thus on human life in general [Eur23a; Eur23e].

Climate change is melting the polar ice sheets, causing sea levels to rise. Extreme weather events are also occurring more frequently, such as heavy local rainfall in some areas and heat waves with droughts in other areas or at other times. Nearly all sectors are affected by the causes of climate change. The European Commission (EC) lists many of them under natural impacts, social hazards, economic hazards, and territorial hazards [Eur23d].

Thus, there is a need to slow down or even stop climate change by achieving climate targets. For this reason, the European Commission has developed several strategies [Log22].

2.1.2 European clean energy transition

The EC has developed different action strategies for different areas. The main objectives for the clean energy transition relevant to this thesis are to build an interconnected energy system with better integration into the grids to support Renewable Energy Sources (RESs), to promote innovative technologies and modern infrastructures, to increase energy efficiency and eco-design of products, to decarbonize the gas sector, and to promote smart integration between sectors. Even more detailed strategies have been developed for additional targets [Eur23b; Log22].

2.1.3 Restructing of the power system

The increasing integration of RESs is restructuring the power system from a few large generation plants to many small ones because RESs, such as Photovoltaic (PV), are small and decentralized. They can be controlled in a decentralized manner and thus supply the consumers themselves on-site [Leh22].

However, conventional power distribution systems have been designed with the assumption that large generation facilities are the only sources of power. The growth of distributed generation, therefore, implies atypical grid usage in the planning sense and can introduce technical issues that must be considered for the efficient and reliable operation of power distribution systems [EGH16].

2.1.4 Emerging power grid management methods

There are new grid management approaches for this. Local energy markets are possible through decentralized plants and a redispatch of these small local grids is necessary [Leh22].

Redispatch is a method of ensuring grid stability, especially in transmission grids. It involves modifying power generation to avoid grid bottlenecks. For example, power plants that were originally scheduled to generate power are downregulated in regions where they would contribute to potential congestion so that lines are not overloaded. At the same time, generators are activated in regions of high consumption to relieve the original congestion. In the first redispatch before the Netzausbaubeschleunigungsgesetz (NABEG), only large conventional plants with a capacity of 10 MW or more were derated. In the Redispatch 2.0 procedure of the NABEG, RES, and Combined Heat and Power (CHP) plants as well as plants with 1100 kW or more that can be remotely controlled by the grid operator are also included in the [BDE22]. As a result, many small plants have to be controlled to ensure grid stability. This large amount of additional control parameters must be managed.

The growing complexity of increasing restructuring and decentralization in the power grid requires increasing digitalization to manage it. This includes the integration of digital monitoring and control systems and intelligent services into the Information and Communication Technology (ICT) infrastructure of the power grid, as well as the use of intelligent metering systems (smart meters), cloud and meters), cloud/edge infrastructures, Big Data and artificial intelligence technologies, and the use of flexibility through Internet-of-Things (IoT) technologies. As a result, Information Technologies(IT) and Operational Technologies (OT) are converging in a way that they can no longer be physically separated. There is also greater interdependence of subnetworks as components potentially depend on different distributed services [Del21].

This leads to new attack vectors through IT bugs and vulnerabilities, e.g., due to a lack of update culture. In 2015, for example, there was a major targeted cyberattack on the power grid in Ukraine that caused a blackout in parts of the country [Leh22].

The energy system is part of the CNI because it is very important for the functioning of society. Therefore, it must meet special security requirements as defined in the BSI [Bun09] and the [Bun21] for Germany.

At the same time, restructuring and digitization trends are making it potentially more unstable, vulnerable, and actively attackable, so it must be robustly secured and made more resilient to successfully avoid blackouts in the future and ensure power security [Del21].

2.1.5 Success of learning systems

Learning systems have achieved remarkable success, especially with DRL, starting with the breakthrough in 2013 with end-to-end learning of the Atari games Mnih et al. [Mni+13] and Double Q-learning [HGS15]. Other successes have been achieved with AlphaGo (Zero), [Sil+17], and MuZero [Sch+20]. DRL involves learning agents with sensors and actuators to achieve specific goals through trial and error. Algorithms such as Twin-Delayed DDPG (TD3) [FHM18], Proximal Policy Gradient (PPO) [Sch+17], and Soft Actor-Critic (SAC) [Haa+18] have proven that they can handle complex tasks.

Due to its success, learning systems are applied in various fields, such as the following.

- In healthcare, Reinforcement Learning (RL) is preferred over traditional Deep Neural Network (DNN) methods to determine the best treatment strategy [NRC20].
- In robotics, RL agents can learn tasks such as pouring water, imitating a human teacher, grasping, balancing balls, and more [NRC20].
- DRL is used in autonomous driving due to its strong interaction with the environment [Sal+17].
- In cybersecurity, DRL is used for automatic intrusion detection techniques and defense strategies [NR21].
- To keep the power grid stable, DRL is used to train defender agents with the Adversarial Resilience Learning (ARL) framework along with attacker agents to recover deviations from a healthy state in an autonomous environment [VWU23].

2.2 Challenges

The power grid essentially consists of generators, consumers, cables, transformers, and other power electronics. It is structured as an amalgamation of several subnetworks and divided into different voltage levels that serve different purposes.

In the power grid, there is the general challenge of grid operation management. This describes how the parameters of the components in the network must be controlled to ensure proper functioning. Specific use cases of grid operation management include frequency and voltage maintenance. In voltage maintenance, the voltage at each level of the power system and the frequency throughout the power system must always be maintained within a certain range to ensure proper operation. If more or less power is fed into the grid, the frequency will increase or decrease. Frequency maintenance in the transmission network is therefore achieved by suitable regulation against deviations from the standard frequency. In distribution networks, voltage maintenance can be achieved by changing the transformer position and by reactive power control [Tur+11] [Leh22].

The trends in the power grid toward the smart grid with, among other things, higher decentralization and digitization are increasing the number of different components and thus controls in the power and coupled ICT network. The complexity of grid operation management is also increasing sharply, as the control of components can have an impact

at various points in the power grid. In addition, the complexity is further increased by the increasing number and interconnectedness of components in the ICT network as well as by decentralized, flexibility-based, and short-term energy markets [MSP+20].

The energy system as a combination of the power grid, ICT network, and energy market represents a Cyber-Physical System (CPS). Classically, such systems are analyzed using monolithic approaches. They are verified, for example, by model checking and extensive simulations to provide a basis for confidence in the CPS [Fis+18]. With increasing complexity, the search space for classical verification becomes too large as the number of stochastic input parameters increases. If the power grid, ICT network, and energy market are considered individually, dependencies and side effects are ignored. Therefore, new approaches are needed to analyze the power system holistically [MSP+20].

One concept for coping with complexity is the cellular approach, an organizational model for energy supply. In addition to digitization, this integrates sector coupling in particular. In this approach, different forms of energy such as electricity, heat and gas as well as mobility, and industrial energy sources and processes are linked with each other in terms of energy technology and economy. By shifting the energy exchange to different forms of energy, the overall energy consumption is to be optimized. The cellular approach is also intended to increase pre-supply safety and quality. In this, there are energy cells that provide the infrastructure for different coupled forms of energy. The energy cells are managed externally by coordinating with their neighboring cells to balance the total generation or consumption across all energy forms [VDE19].

Since it will no longer be possible to analyze and thus control the power system manually due to the increasing complexity, the vision is to achieve fully automatic, autonomous operation control in addition to mastering the complexity [Vei22].

A new approach for validated resilient network operations is the ARL approach. In this approach, there are multiple agents, namely an attacker and a defender, but it is not considered a Multi-Agent System (MAS) because the agents neither share a common goal nor communicate directly. The attacker agent attempts to destabilize the CPS, and the defender agent attempts to keep it in a stable state. Both operate on the same simulated power grid, so their actions affect the grid and therefore each other's observations. The purpose of the attacker agent is to train the defender agent's model specifically for resilient operational strategies. Therefore, the attacker agent can be viewed as a complex sampling helper for the defender agent that can learn better by potentially covering a larger portion of the search space in the samples. A unique feature of the ARL approach is that the agents are not explicitly given each other's actions, so the models are trained in a more realistic scenario. The goal is to train as new attack and defense strategies as possible to evaluate, understand, and thus improve security in advance [Fis+18].

The ARL approach is based on the autocurriculum hypothesis, according to which agents generate dynamics in part through competition and cooperation. This hypothesis is modeled on evolution, where different entities also interact and adapt. According to this hypothesis, innovation occurs when parts of the system are pushed by others into unknown search space regions where previously applied solutions no longer work and therefore new ones must be developed. This process is called autocurriculum because it occurs automatically in cycles, in

that new solutions in turn raise new challenges that must be solved. As the challenges become more complex over time, innovation increases even further [Lei+19; Bak+20].

ARL agents are competitors and learn opposing strategies in a cyclic competition with opposing goals. In doing so, they implicitly learn each other's strategies and adaptively develop their strategies to not only compensate for the behavior of the competing agents, but to push them back as far as possible to dynamically approach their goal.

The agents can be successfully trained using DRL. In [VWU23; Vei+22], a successful attack on a power grid with reactive power control from Distributed Energy Resources (DERs) by ARL agents based on DRL is demonstrated for this purpose. In general, Artificial Neural Networks (ANNs) can be used alongside other models in DRL because they can learn incrementally and are powerful. They can be complex, which is why they make potentially incomprehensible [PV20a] predictions and thus decisions. Moreover, they are used to minimize a cost function and thus only indirectly learn a model of the problem [Fis+18]. Learned strategies have therefore been difficult to reproduce with classical supervised learning of ANNs.

However, the technologies to be used for the vision of fully automated operations management must be trustworthy to be accepted and deployed, since the error bounds for an adaptive learning system in the CNI environment are much tighter than in standard benchmark environments. Errors must be avoided much more in this context, as they can have potentially catastrophic consequences [Vei22].

There are also some more general regulations on components in CNI by the Bundesamt für Sicherheit in der Informationstechnik (BSI) [Bun09] and the IT Security Act 2.0 [Bun21]. According to "§ 9b Untersagung des Einsatzes kritischer Komponenten" (3), critical components may only be used if the manufacturer has issued a declaration of confidence to the operator of the critical infrastructure. The declaration of confidence must state how the manufacturer ensures that the critical component does not have any technical features that could reduce the security, confidentiality, integrity, availability or functionality of the critical infrastructure and, in particular, could be misused for sabotage, espionage or terrorism.

Although this is aimed more at the threat of adversary use of IT components, the Declaration of Trust indirectly states that the manufacturer must also ensure that the components do not harm the CNI.

Agents trained using DRL methods, for example, promise true resilience. However, unlike intrinsically interpretable DRL models [PV20b], no guarantees can yet be given for the learned black-box models. For operators of CNIs this is a necessity, since especially in such critical or very critical areas as CNI no responsibility can be taken for an unknown and unverifiable control system.

In particular, the risk that the DRL models themselves become unknown unknowns must be minimized so that their learned behavior can have harmful effects only with sufficient probability. The unknown behavior of an ARL agent is known, its model parameterization should be attempted to be accurately analyzed so that the behavior can be understood and appropriate countermeasures can be taken. However, there may also be an unknown behavior of an agent that is not known. According to [RB14], an unknown event raises the question of whether or not it could have been predicted. The question is whether a possible analysis before the occurrence of an event would have predicted or uncovered a previously unknown

behavior that could then have been further analyzed to possibly still take countermeasures. When an unknown event cannot be predicted and has catastrophic consequences, it is also referred to as a Black Swan. In such a case, one should not be deceived in advance by false predictions [Inv22], because such events are unknown and can therefore potentially always occur.

Even a robust system cannot withstand black swans because they are not predictable a priori. Therefore, a resilience approach is used that can dynamically adapt to the environment and thus respond to long-term unpredictable events and specifically repair the system after a failure.

To ensure that a DRL model does not unpredictably lead itself to unknown agent behavior, it must be analyzed and explained in such a way that the learned behavior is predictable enough to minimize risk due to such a potential hazard factor.

If agents with learned black-box models are to be used in CNI, it is necessary to provide guarantees for them, as they have the potential to significantly compromise the security of the overall system. Without guaranteed behavior of agents, operators cannot take responsibility for such an unknown and unverifiable control system. In [Vei23],, an architecture is presented that is intended to provide such guarantees and is suitable for use in CNI such as the power grid.

2.2.1 Transparency of agents' learned strategies

Agents operating in complex environments, such as complex networked systems, potentially face many different situations and learn complex behaviors to cope with them according to their goals. To understand how agents achieve their goals, the effects of their strategies are studied in terms of rewards or effects on the environment. One such example can be found in [VWU23], where ARL agents are deployed to cause voltage band violations in a power grid. They achieve this goal by exploiting a vulnerability in the deployment of voltage regulators in the network used. How the actual attack works is analyzed by examining the impact of the attacker's actions on the victim buses. This is sufficient for commonly observed behaviors, but it is not deeply interpreted and there is no guarantee that the extracted strategy explained by the investigations will be used for all situations, i.e., for all observations from the environment. This is especially important when dealing with control agents that are expected to achieve a goal in all possible situations, e.g., defender agents, for which there is the even greater problem of coping with an infinite horizon in the explanation.

This results in the need to make agents' learned strategies transparent, i.e., to accurately describe their behavioral model by a more comprehensible model. This leads to the trade-off of wanting to construct powerful learning systems, i.e., relying on DRL, on the one hand, but also being able to explain them afterwards by more comprehensible models such as DTs [LV23a].

2.3 Research question and hypothesis

This leads to the following research question:

> How to increase the explainability of the models used, and thus of the FF-DNN policies trained with DRL methods in the ARL approach, to minimize the likelihood of potentially catastrophic failures occurring, i.e. minimize the risk that ARL agent behavior will lead to harmful effects in cases where it would otherwise be unknown, to increase confidence in future DRL-based automatic control systems for the operation of CPS networks and, in particular, the power grid?

The author hypothesizes that the explainability of an ARL agent model can be increased by methods that simplify the learned input-output mapping with the parameters of the model within its architecture to better understand how an ARL agent obtains an action as an output for observation as an input. In doing so, it is important to use the exact input-output mapping of the DNN model used to later accurately validate whether the agent has learned strategies with harmful effects in subsets of input-output mappings that rarely occur and would not be known without such a method. Approximation methods cannot accurately validate this. Excluding harmful effects in agents' learned strategies minimizes the likelihood of potentially catastrophic errors caused by the agents' purely internal control mechanism.

2.4 Outline

The thesis is structured as follows: First, relevant basics of the power grid, Machine Learning (ML), Deep Learning (DL), RL, DRL, eXplainable Reinforcement Learning (XRL), and Satisfiability Modulo Theories (SMT) are presented. Section 5.1 provides requirements based on Section 2.2. The XRL methods presented in the overview in Section 3.6 and in detail in Section 4.1 are analyzed in Section 5.2 to determine whether they meet the requirements, and the research gap is identified. Based on these and other not-quite-fitting-but-still-supporting methods from related work in Chapter 4, contributions to fill the research gap are explained in Section 5.3. In addition, the developed method, the NN2EQCDT algorithm, is conceptually presented in Chapter 6. In Chapter 7, a power system scenario is described that is used for the application and thus the evaluation of the NN2EQCDT algorithm in Chapter 8. Finally, Chapter 9 concludes the thesis and describes possible future work for further improvements.

3 Basics

3.1 Power grid

3.1.1 Criticality of electrical energy

Electrical energy is the highest form of energy. It can be converted into other forms of usable energy with minimal losses. Since it can be used flexibly to drive motors, it can replace human actions. Therefore, the high availability of electrical energy is critical to our modern standard of living. The damage caused when electrical energy is unavailable is very high. In the event of a power outage, lights go out, the heating stops working, data on computers can be lost, elevators come to a standstill, traffic lights no longer regulate traffic, stores close because cash registers stop working, banking services are no longer available, gas stations stop dispensing gasoline because pumps stop working. In the event of a medium-length outage, food can spoil in the refrigerator, which is a problem in the long run because bakeries and butchers can no longer produce new food either. However, even greater economic damage is caused by interruptions in production, e.g. in automobile or chip manufacturing [Sch12].

3.1.2 Architecture

In steady state, the power grid consists of three main categories of components: Overhead lines, underground cables, and transformers. This is modeled in the π-equivalent circuit diagram, as shown in Figure 3.1. Overhead lines are used primarily in transmission networks, while underground cables are used primarily in distribution networks in Germany.

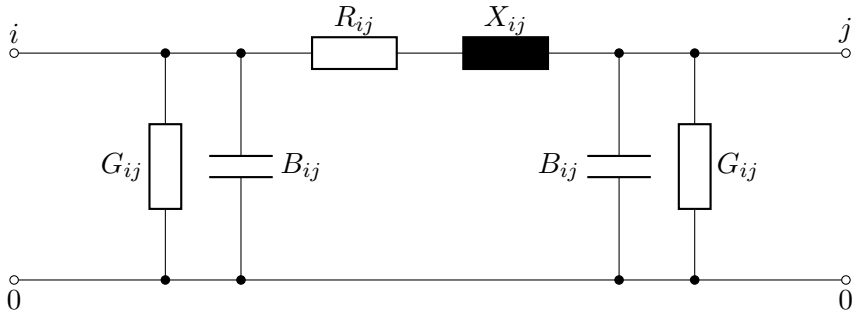


Figure 3.1: π-equivalent circuit diagram, from [Leh22] for modeling the operating equipment of underground cables and overhead lines, cfg. also [GS94]

Ground cables are modeled as conductances G_{ij} and susceptances B_{ij}. They have strong insulation from the environment to minimize leakage currents, resulting in small conductances. Since underground cables consist of several closely spaced wires that act like capacitors, their susceptance values are higher. In overhead lines, the wires are not insulated and have a high wire diameter, so their resistances R_{ij} are low. The wires are further apart compared to underground lines, so their reactances X_{ij} are lower.

All major power systems use Alternating Current (AC) instead of Direct Current (DC) because it has some advantages. Most importantly, with the invention of the AC transformer,

electricity can be transmitted over long distances at high voltages. This is advantageous because the power dissipation is proportional to the square of the current, i.e., $P_{loss} \approx RI^2$, and the current can be reduced by using high voltages because $P = UI$. Thus when high voltages are used, the power dissipation is reduced so that power can be transmitted efficiently over long distances using AC.

In a power grid, there are different constant voltage levels, ranging from ultra-high voltage, high voltage, medium voltage to low voltage, each serving a different purpose. They are interconnected by transformers of different sizes that convert the different voltage levels into each other. However, the frequency of the power grid is nominally 50 Hz everywhere in the UTCE, the European power grid [Leh22].

All quantities such as voltage, current, and resistance are expressed as *per unit* of a base or reference value. The unit value for all quantities is defined as the ratio of the quantity to its base as a decimal number. Since both the percentage and unit value methods are simpler and more meaningful than actual values, they are preferred. The unit value method also has the advantage over the percentage value method of being closed in multiplication, so it is preferred [GS94].

3.1.3 Network modeling

AC voltages and currents are described in the time domain by:

$$u(t) = \hat{u}sin(\omega t + \varphi_u) \tag{3.1}$$

$$i(t) = \hat{i}cos(\omega t + \varphi_i)\ . \tag{3.2}$$

By integration over the power in the time domain, the effective power and the voltage can be calculated with:

$$p(t) = \frac{\hat{u}^2}{R} sin^2(\omega t) \tag{3.3}$$

$$P_{eff} = \frac{\hat{u}^2}{2R} \tag{3.4}$$

$$U_{eff} = \frac{\hat{u}}{\sqrt{2}}\ . \tag{3.5}$$

By extending into the frequency domain, the measures introduce phasors in the steady state, which are used in another time-independent consideration:

$$\underline{u}(t) = \hat{u}(cos(\omega t + \varphi_u) + jsin(\omega t + \varphi_u)) = \hat{u}e^{j(\omega t + \varphi_u)} = \hat{u}e^{j\varphi_u}e^{j\omega t} = \underline{U}e^{j\omega t}\ . \tag{3.6}$$

The complex power is then given by:

$$\underline{S} = \underline{U}\underline{I}^* = \hat{u}\hat{i}e^{j(\varphi_u - \varphi_i)} = \hat{u}\hat{i}e^{j\varphi} = Se^{j\varphi} \ . \tag{3.7}$$

On the other hand, the apparent power S is composed of the active power P and the reactive power Q:

$$\underline{S} = |S|\, cos(\varphi) + j|S|\, sin(\varphi) = \hat{u}\hat{i}cos(\varphi) + j\hat{u}\hat{i}sin(\varphi) = P + jQ \ . \tag{3.8}$$

The various measures of resistance $R = \frac{\rho(T)l}{A}$ as a function of temperature ρ, line length l and cross-section A, reactance X, conductance G, and susceptance B have already been mentioned above in the description of the π equivalent circuit in Section 3.1.2. They are defined by the impendence Z and admittance Y of the complex phase conductors already described:

$$\underline{U} = \underline{Z}\underline{I} = (R + jX)\underline{I} \tag{3.9}$$

$$\underline{I} = \underline{Y}\underline{U} = (G + jB)\underline{U} \ . \tag{3.10}$$

The admittance,

$$\underline{Y} = \frac{R - jX}{R^2 + X^2} \ , \tag{3.11}$$

is used to construct a nodal admittance matrix, since it can be used to calculate the nodal values:

$$\underline{I}_{ik} = \underline{Y}_{ik}(\underline{U}_i - \underline{U}_k) \ . \tag{3.12}$$

The details are omitted here.

The general power flow equation can then be derived as follows:

$$\underline{S}_i = \underline{U}_i \sum_{k=1}^{n} \underline{Y}_{ik}^* \underline{U}_k^* \ . \tag{3.13}$$

With static and equipment-specific admittances, measured nodal powers, information about the nominal power of consumers or generators as PQ nodes or PU nodes, and estimated nodal voltages as initial values, the actual nodal voltages and thus currents can be approximated with mostly iterative algorithms that solve this nonlinear equation. Concrete solution algorithms such as the Newton-Raphson method or the Gauss-Seidel method are not described further here [Leh22; GS94].

3.1.4 Power quality

Since electrical energy is critical, as described in Section 3.1.1, various quality measures are required to maintain its availability. An important condition for safe operation is the (n-1) criterion in power system design and operation. It essentially conditions the network structure in such a way that, in the event of a line failure, the load can be distributed immediately to avoid permanent limit violations of operating voltages, voltage bands, and short circuits, as well as equipment loads that could endanger the network. Supply interruptions, sequential tripping by other protective devices that were not directly affected by a fault, loss of stability of generating units and the possibility of changing or interrupting transmissions must also be prevented with this criterion [Leh22].

3.1.5 Supply quality

Ideally, each node should have the specified nominal voltage, but this is not possible because the voltage drops along the lines. This voltage decrease with symmetrical line characteristics can be calculated with

$$\underline{Z} = R + jX \tag{3.14}$$

$$\underline{I}_i = \underline{I}_j = \underline{I} = I_W - jI_B \tag{3.15}$$

$$\Delta\underline{U} = \underline{ZI} = (R + jX)(I_W - jI_B) = I_W R + I_B X + j(I_W X - I_B R)\ , \tag{3.16}$$

which is described along the π-equivalent circuit diagram in Figure 3.1 [Leh22].

Therefore, a permissible voltage range is defined that still allows functional operation [Sch12].

According to the description in [Mar04], DIN EN 50160 for Supply Voltage Characteristics (SVC) for public distribution networks specifies the requirements for voltage parameters and specifies their permissible deviation ranges at the common coupling points of customers in public Low Voltage (LV) and Medium Voltage (MV) electrical distribution networks, under normal operating conditions.

LV networks require that the phase-to-phase nominal RMS voltage does not exceed 1000 V and MV networks require that it is in the range [1 kV, 35 kV]. However, at the current voltage level, only small deviations and changes in voltage and frequency are allowed, as shown in Figure 3.2, otherwise, the power system may become unstable and collapse [Leh22].

In LV and MV networks, the voltage level may deviate only 10 % from the nominal voltage level for at least 95 % of the week. Thus, on average for 10 min, there may be other instabilities such as dips in the supply voltage or short interruptions in the supply voltage, which are not further described here [Mar04]. This general voltage range can also be represented in the unit system as [0.9 p.u., 1.1 p.u.].

In addition to the voltage quality described, there are other criteria for general supply quality such as frequency constancy, voltage drops, flicker, voltage symmetry [Sch12], etc., but these are not described further.

Parameter	**Supply voltage characteristics according to EN 50160**
Power frequency	LV, MV: mean value of fundamental measured over 10 s ±1 % (49.5-50.5 Hz) for 99.5 % of week −6 %/±4 % (47-52 Hz) for 100 % of week
Voltage magnitude variations	LV, MV: ±10 % for 95 % of week, mean 10 min rms value

Figure 3.2: Power frequency and voltage magnitude variations according to EN 50160, from [Mar04]

3.1.6 System operators

There are two types of network operators: Transmission System Operators (TSOs) and Distribution System Operators (DSOs). TSOs have the task of ensuring energy balancing in their network in Europe. DSOs, on the other hand, are responsible for the optimal operation of smart distribution grids. For example, they have to balance the reactive power in their grids while respecting the permissible voltage bands. This must be considered together since the voltage level depends directly on the active and reactive power contributions. The network operators perform their tasks in this respect with the help of auxiliary services.

For example, TSOs use grid frequency control. It balances demanded and supplied power by controlling generation or consumption units of different sizes to stabilize the grid frequency at 50 Hz [Leh22].

DSOs use various Active Network Management (ANM) techniques to meet network constraints. Most involve active voltage regulation and active power flow management, but there are also systems that use reactive power, such as coordinated voltage regulation, which uses reactive power control from DERs [EGH16].

According to VDE-AR-N 4110:2017-02 [VDE17], SVC in the medium-voltage level, generators can be permanently connected to the grid if the agreed supply voltage and frequency in steady state are in the orange range in Figure 3.3. However, if they are in the pink or blue range, they must still be synchronized with the grid and respond to control commands for at least the specified time, and only then may they disconnect from the grid for self-preservation.

In addition to static constraints on voltage and frequency values, VDE-AR-N 4110:2017-02 also imposes constraints on frequency and voltage gradients, as shown in Figure 3.4. According to this, the voltage shall not change by more than 5 % of the agreed supply voltage U_c per minute, unless the voltage range deviates by only ±10 %U_c. The frequency shall not change by more than 0.5 % of the standard frequency f_n per minute.

3.1.7 Power flow analysis

In the past, the calculation of power networks and lines in steady state was performed with analog network models on analog computers. Nowadays, since the advent of digital computers,

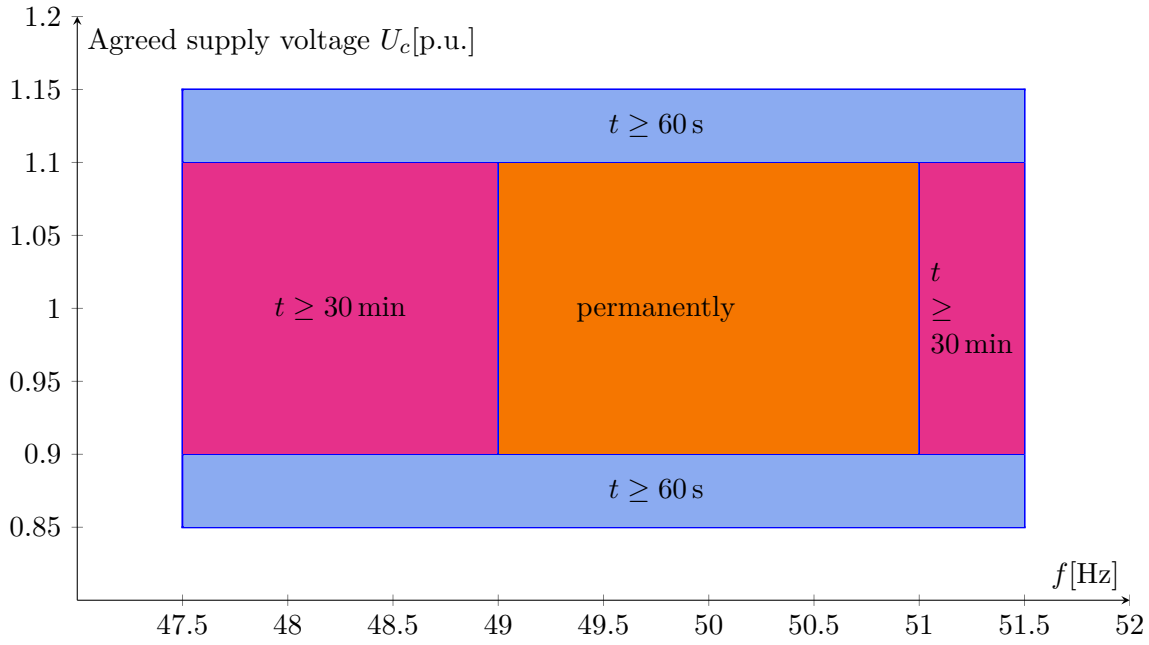


Figure 3.3: Requirements for the operation of generators in steady state according to VDE-AR-N 4110:2017-02, SVC at medium voltage level, by [VDE17]

Range	**Voltage gradient**	**Frequency gradient**
General	$< 5\,\% \frac{\mathrm{U_c}}{\mathrm{min}}$	$< 0.5\,\% \frac{\mathrm{f_n}}{\mathrm{min}}$
$[90\,\%\mathrm{U_c}, 110\,\%\mathrm{U_c}]$	$> 5\,\% \frac{\mathrm{U_c}}{\mathrm{min}}$	

Figure 3.4: Frequency and voltage gradient constraints at steady state according to VDE-AR-N 4110:2017-02, SVC at medium voltage level, from [VDE17]

it is calculated with Power Flow Calculation (PFC), or analysis, as described in Section 3.1.3. It enables optimal design in planning and optimal power flow in operation according to economic and technical criteria [Sch12]. The solution of PFC can be performed with different algorithms, as shown in Section 3.1.3 [Leh22].

3.1.8 Voltage control by reactive power control

The voltage can be controlled with reactive power injection. This ANM scheme is used with PV because their reactive power injection can be controlled by setpoints.

Figure 3.5 shows a schematic representation of a radial network. Here P_j and Q_j represent the active and reactive power flowing downward from node j. P_0 and Q_0 represent the power flow from the external network connected to a substation. p_j and q_j correspond to the power outflow from bus j, where (c) is the superscript for the consumption and (g) is the superscript for the generation.

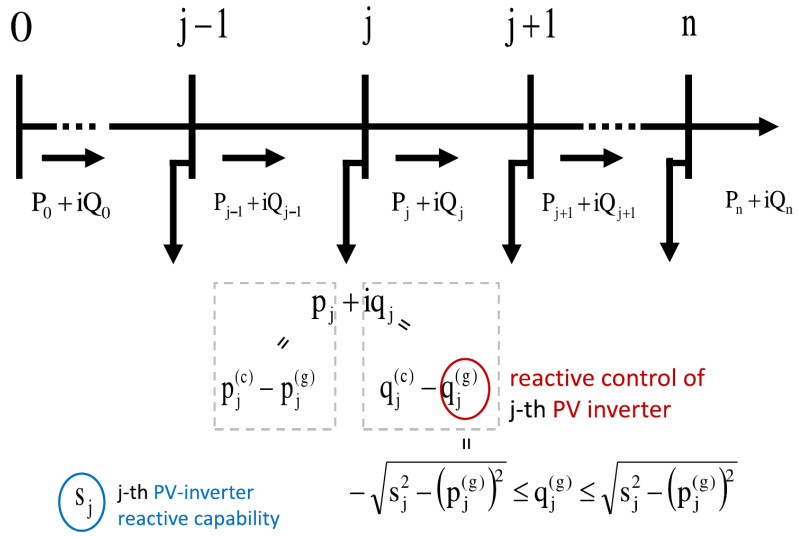


Figure 3.5: Diagram and notations for the radial network, from [Tur+11]

The *LinDistFlow* equations [BW89a] [BW89c] [BW89b] for the radial network, as shown in Figure 3.5,

$$P_{j+1} = P_j - p_{j+1}^{(c)} + p_{j+1}^{(g)} \tag{3.17}$$

$$Q_{j+1} = Q_j - q_{j+1}^{(c)} + q_{j+1}^{(g)} \tag{3.18}$$

$$V_{j+1} = V_j - \frac{r_j P_j + x_j Q_j}{V_0} \ , \tag{3.19}$$

are used to regulate the voltage by controlling the reactive power injection of the inverter PV.

The generated reactive power is limited by:

$$\forall_j : \left| q_j^{(g)} \right| \leq \sqrt{s_j^2 - (p_j^{(g)})^2} \equiv q_j^{max} \tag{3.20}$$

where s_j is the apparent power capability of the inverter, which in turn is limited by $\max p_j^{(g)}$ [Tur+11].

The reactive power capability is based on the active power and the size of a PV inverter. PV inverters can be overrated by, for example, 10 % as shown in (b) of Figure 3.6.

Their maximum active power that can be injected into the grid is then limited, but they can still inject reactive power according to their overrating as shown in (a) of Figure 3.6, even when operating at full power generation [Liu+08; LB08] [KHM14].

According to the LinDistFlow equations, if the apparent power s_j is greater than the generated active power $p_j^{(g)}$, then an inverter at bus j can supply or consume reactive power

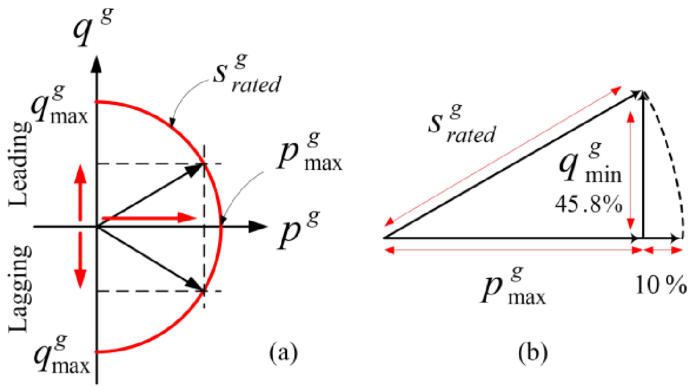


Figure 3.6: PV inverter capability curve for (maximum) reactive power injection as a function of active power (a) and inverter size (b), from [KHM14]

$q_j^{(g)}$. This provides a mechanism for fast voltage regulation because it follows from the LinDistFlow equations that the voltage at the next bus $j+1$ is different from the voltage given by $q_j^{(g)} \sim Q_j \sim -V_{j+1}$ depending on the reactive power generated.

In Figure 3.7 a reactive power control function, also known as Volt/VAR, corresponding to a simplified version from [Ele10], is presented [ZL16; APE21].

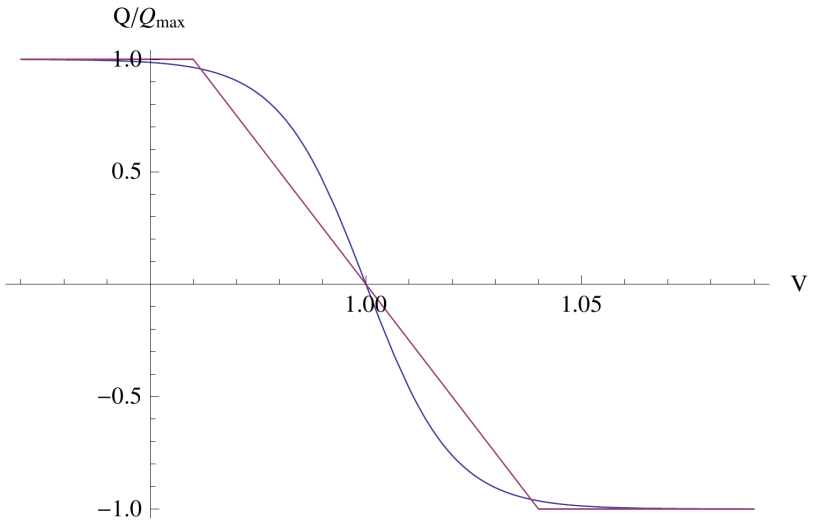


Figure 3.7: The proposed simple $q_j^{(g)}$ control function from [Ele10] (dark red, piecewise linear curve) and a very similar control function defined to improve the convergence properties of an AC solver (not considered further) (blue, smooth curve) from [Tur+11]

Besides this, [ZL16] proposes a local voltage control based on limited reactive power injection capacity. With $\alpha(t) \coloneqq 1$, $\forall_j : c_j(t) \coloneqq 0, \mu_j(t) \coloneqq 1$ the equation (20),

$$q_j(t+1) = [1-\alpha(t)]q_j(t) + \alpha(t)\mathbb{P}_j[(1-d_j c_j)q_j(t) - d_j(V_j(t)-\mu_j)]\ \forall_j\ , \tag{3.21}$$

reduces to the simplified and vectorized version

$$\boldsymbol{q}^g(t+1) = [\boldsymbol{q}^g(t) - \boldsymbol{D}(\boldsymbol{V}(t) - 1)]^+ \,, \tag{3.22}$$

as used in [JL18] and [VWU23], where $\mathbb{P}_j[\cdot]$, resp. $[\cdot]^+$ is a projection (at bus j) of invalid values onto the interval $[\underline{q_j}, \overline{q_j}]$, respectively $[\underline{\boldsymbol{q}^g}, \overline{\boldsymbol{q}^g}]$, i.e., on the feasible range of setpoints for $\boldsymbol{q}(t+1)$. The diagonal matrix $\boldsymbol{D}$ is composed of step sizes.

3.2 Machine Learning

ML refers to the problem of automatically learning optimal decisions. In this process, optimality is supposed to increase with time. It can be divided into supervised, unsupervised, and RL. In supervised learning, the assignment of inputs to outputs is optimized using a set of example pairs. These problems include text or image classification, regression problems, and sentiment analysis. Common to all supervised methods is that they learn responses from the ground truth data provided as training data. In doing so, most methods assume that the data are independent and identically distributed (iid) to learn their underlying characteristics.

In contrast, in unsupervised learning, there is no ground truth data against which a model can be optimized. The idea is to learn the hidden structures of a given dataset itself. An example of this is clustering a dataset by combining elements into separate groups of clusters with certain relationships that all elements in a cluster have, but that distinguish clusters from others.

RL differs from these two static learning paradigms because it addresses the problem of automatic dynamic learning from prior state-action pairs of optimal decisions over time [Lap20].

3.2.1 No-freee lunch theorem

Learning theory states that ML algorithms can train models to generalize well from finite training data sets. However, in logic, inductive reasoning, i.e., deriving general rules from only one part, is not allowed, but only using the entire set.

ML gets around this problem by not returning very specific rules as output, but rules that are only likely to be correct for most members of sets.

But which algorithm has the least error or is the "most correct", i.e. the best? Unfortunately, there is no No-Free Lunch (NFL) in ML. This theorem states that every classification algorithm has the same error rate when classifying test data, averaged over all possible data generation distributions. This means that no ML algorithm is universally better than another, i.e., for every problem [Wol96].

But which algorithm should then be chosen for predicting values? Fortunately, the NFL theorem only holds when averaging over all possible data-generating distributions. Therefore, ML algorithms can or must be developed with assumptions about distributions in certain real-world applications to achieve good results [GBC16].

3.3 Deep Learning

DL uses computational models to learn representations of data. With multiple levels of processing, they can learn multiple levels of abstraction.

Many ML problems become extremely difficult when the number of dimensions in the data is high because the number of possible different configurations of variable sets increases exponentially with the number of variables. This phenomenon is known as the *curse of dimensionality* because additional dimensions, and thus usually more data points, do not necessarily mean that a more accurate model can be trained because the number of possible configurations is usually much larger than the number of training data [GBC16].

But DL methods can solve this problem better. It has successfully shown that it can discover complicated structures in large datasets with high dimensions. Therefore, it is applicable in many fields.

Traditional ML techniques, as described in Section 3.2, are limited in their ability to process raw natural data. For each problem, engineers must manually adjust the model with significant expertise. Nowadays, DL is mostly used instead, as it can be fed with raw data. It can automatically discover representations needed for recognition or classification.

Models in DL consist of nonlinear modules that convert representations between layers into higher-order representations. They can learn very complex functions. In this process, the parameters of the layers are not designed manually by human engineers but are learned from the provided training data using universal learning methods. These use the backpropagation algorithm to compute gradients that are used to change the internal parameters of the model layers according to the connections to the previous layers and the optimization algorithm. DL not only has the advantage of requiring very little manual design but can also take advantage of the increasing amount of computation and data available to create more accurate models [LBH15].

3.3.1 Supervised learning

In supervised learning, the training data consists of input-output samples from which a function is approximated by updating a model. Therefore, the error between the actual and desired output values is calculated by an objective function. The goal is to reduce the error for all predictions. For this purpose, the internal adjustable parameters, also called weights, of the model are changed [LBH15].

3.3.2 Neural Network Models

The simplest form of a DNN is an FF-DNN or MultiLayer Perceptron (MLP). Its goal is to approximate a function f^* by learning the value of the parameters $\boldsymbol{\theta}$ that best approximates the function with $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$. The network consists of several linear layers through which the input data is propagated sequentially.

Such linear models are limited to the approximation of linear functions. To extend DNN to represent nonlinear functions of the input, the input of a layer is generally preprocessed by a

nonlinear transformation. This nonlinear transformation can be done in a variety of ways, but in DL it is learned. Most DNNs use an affine transformation whose parameters are learned, followed by a fixed nonlinear function, an *activation function*.

In the case of linear transformation $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}$, the controllable parameters are the weight matrix $\boldsymbol{W}$ and the bias vector $\boldsymbol{b}$, which are learned. It becomes a hidden unit of DNN by applying an element-wise nonlinear function to the output of the linear transformation $g(\boldsymbol{z})$.

These units are then stacked in layers, something like this:

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \tag{3.23}$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right) . \tag{3.24}$$

According to the *Universal Approximation Theorem* [HSW89], a linear output layer and at least one hidden layer with an arbitrary activation function, can approximate any measurable *Borel function* from a finite-dimensional space into another space. With a sufficient number of hidden units, an FF-DNN is theoretically capable of approximating a function from data with an arbitrary nonzero error amount. Any continuous function on a closed and bounded subset of $\mathbb{R}^n$ is *Borel-measurable*. However, the concept of Borel-measurability is not further described here, since this explanation is sufficient for a clear understanding.

Although FF-DNNs can represent features of provided data, there is no guarantee that training algorithms will be able to learn them. To improve this DNNs can also use a variety of other layers, components, and architectures, such as convolutional layers and residual layers, skip connections, and dropout, but these are not described further [GBC16].

3.3.3 Activation functions

Activation functions are used to extend linear models to represent nonlinear functions in DNN, as described in Section 3.3.2. These are fixed nonlinear functions that are usually applied element-wise to the input matrices.

A popular choice for an activation function is to use Rectified Linear Unit (ReLU) defined by [GBC16]:

$$\mathrm{ReLU}(x) = \max\{0, x\} \begin{cases} x & , x \geq 0 \\ 0 & , x < 0 \end{cases} . \tag{3.25}$$

In particular, their use has been observed to significantly improve the performance of object recognition systems [Jar+09; NH10]. [GBB11] shows that rectifying neurons are also better models of biological neurons than tanh and thus sigmoid neurons. The function ReLU has hard nonlinearity and nondifferentiability at zero, which may initially be seen as a drawback, though these turn out to produce sparse representations with real zeros that seem remarkably suitable for approximating naturally sparse data. Other activation functions, such as the tanh or sigmoid functions, are not described further here.

3.3.4 Optimization

With the extension of linear models by activation functions, as described in Section 3.3.2, the loss functions of the then nonlinear DNNs become nonconvex. Therefore, DNNs are trained with iterative gradient-based optimizers or convex optimization algorithms to reduce the error, rather than direct solvers as used for linear equations. Convex optimization algorithms have global convergence guarantees but in practice exhibit numerical instabilities [GBC16].

For the weight adjustment described in Section 3.3.1, gradients are calculated for each weight. These indicate the amount by which the error would change if the weights were changed by a tiny amount. Thus, the weights are adjusted in the direction opposite to the gradients to minimize the error. The objective function can be thought of as a hilly landscape in a high-dimensional space of weighting values. The negative gradient indicates the direction of the steepest descent, so the update step brings the function closer to the minimum with a lower average output error [LBH15].

3.3.4.1 Backpropagation

The backpropagation algorithm [RHW86], as mentioned in Section 3.3, is the basis of optimization algorithms for DNN. It computes the gradients of the weights according to the specified loss function and the target labels, i.e., it iteratively computes derivatives to the target labels [LBH15].

The analytical calculation of gradients is simple, but the numerical evaluations can be computationally intensive. The backpropagation algorithm is simple and inexpensive for the numerical evaluation of gradient calculations [GBC16].

As shown in Figure 3.8 on the right, the input must first be propagated forward through the DNN model to compute the output values y_l.

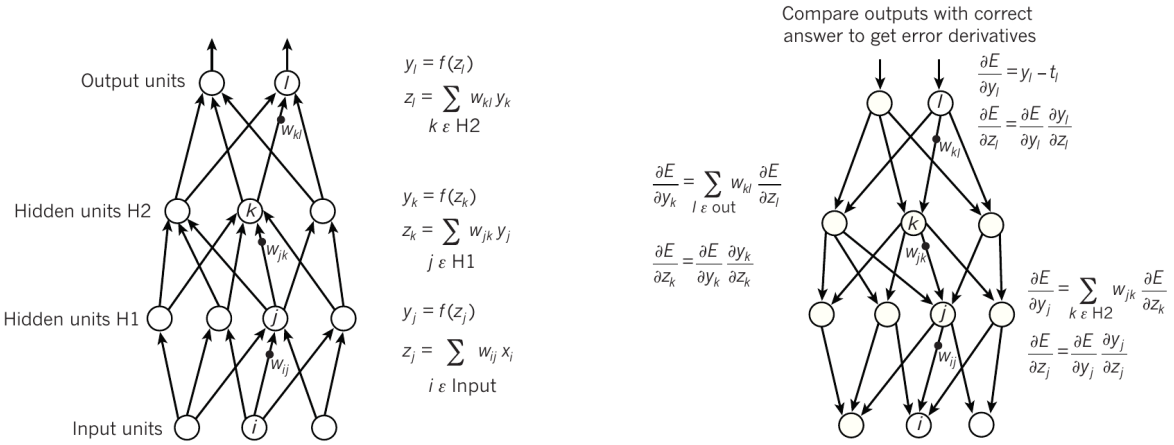


Figure 3.8: Backpropagation of DNN, the left side shows the forward pass and the right side shows the backpropagation, from [LBH15]

These are then used to calculate the loss with the difference between the predicted and true output, seen on the right side of the backpropagation. The difference, however, also represents the final derivatives of the error from the predicted values. Taking advantage of the chain rule,

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \ , \tag{3.26}$$

the gradients are then further computed iteratively for all weights corresponding to the previous [LBH15; GBC16].

3.3.4.2 Optimization algorithms

For optimization of DNNs, algorithms such as Stochastic Gradient Descent (SGD) or Adam [LBH15] are used in practice. The SGD optimization algorithm applied to non-convex loss functions of DNNs as described in Section 3.3.4 has no convergence guarantee and is sensitive to the initial parameter values [GBC16].

More concretely than in Section 3.3.4.1, the cost function with the sum over the training examples can be written as a negative conditional log-likelihood:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{x,y\sim\hat{p}_{data}}\mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \ . \tag{3.27}$$

The update procedure of SGD is performed on small minibatches $\mathbb{B} = \{\boldsymbol{x}^{(1)} \dots \boldsymbol{x}^{(m')}\}$. It uses the estimation of gradients,

$$\boldsymbol{g} = \frac{1}{m'}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{m'}\mathcal{L}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \ . \tag{3.28}$$

previously computed using the backpropagation algorithm as described in Section 3.3.4.1.

The actual updating of the weights by SGD is then performed as follows, using the hyperparameter ϵ as the learning rate:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\boldsymbol{g} \ . \tag{3.29}$$

Other optimization algorithms, such as Adam, or concepts like regularization are not described further [GBC16].

3.4 Reinforcement Learning

RL addresses the problem of automatic dynamic learning of optimal decisions from prior state-action pairs over time, as described in Section 3.2. It is inspired by the [Nob57] trial-and-error paradigm of [Qin+22], which states that humans can learn without guidance from

others just by interacting with an environment. In doing so, they learn from their accumulated experiences. This includes the aspects of causes and effects, results of actions, and how to achieve a goal in an environment [Qin+22].

In RL, agents are formally deployed in an environment and attempt to achieve a goal, also called an objective. They do this by observing the environment, or usually part of it, and performing actions, as shown in Figure 3.9. There are two types of actions in an environment, discrete and continuous, which are used depending on the use case.

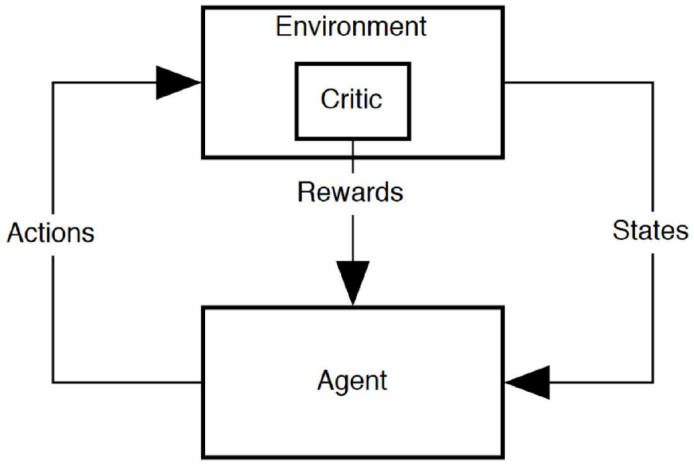


Figure 3.9: RL Architecture [PV20b]

Agents assume that an action they perform will move them further from their state toward their objective. How well an action moves an agent from its state to its objective is measurable by a scalar-valued reward that the environment assigns to the agent. The environment assigns such a reward value according to the action taken by an agent in its state and the agent's objective.

There are different reward systems. For example, the environment may reward the agent regularly or often, at each interaction, or once (sparse rewards). A reward is intended to reinforce an agent's behavior because it is a measure of how well the current state-action pair matches the objective. Thus, an agent seeks to achieve the largest cumulative reward, analogous to the reward process of the brain's dopamine neurons [Sta+16]. In this process, all rewards are local, because the same action is potentially not always rewarded in the same way. After all, the (agent in the) environment may change its state, leading to a different reward for the same action. Therefore, observations are also necessary to infer the state of an agent in the environment. The complete state of the agent (in the environment) cannot be specified because it is usually either not possible to obtain all the information or the measurements contain noise. Observations, i.e., measured parts of the environment based on the agent's previous action, should therefore inform the agent as best as possible about its state in the environment [Lap20].

3.4.1 Complications

RL also depends on the iid property of the training data, as in supervised learning as described in Section 3.2, since agents may otherwise receive incorrect impressions if, for example, there are many data points in one region of the search space and other areas are not covered.

A second complication for agents is the tradeoff between exploration and exploitation. Agents should, on the one hand, exploit, i.e., optimize, their already acquired knowledge, and, on the other hand, actively explore the environment to potentially achieve fundamentally better results. However, too much exploration can seriously degrade results, and agents may also forget already acquired knowledge that potentially works well.

A third complication concerns the timing of rewards. When agents receive only highly delayed rewards, as in the case of sparse rewards mentioned above in Section 3.4, they may have more difficulty detecting causalities because influences on these rewards are only implicit and aggregated [Lap20].

3.4.2 Markov family

A Markov Process (MP) requires a finite set of states and the Markov property of an observable system that changes state only according to certain dynamical laws. The observations of the system are a sequence of states forming a chain. The Markov property is given if the future system dynamics of any state depend only on that state, that is, states are distinguishable and unique to each other. The dynamics of MP are thereby modeled by probability-based transitions between states.

By adding a scalar number as a reward for each transition and a discount factor γ (gamma), the MP is extended to a Markov Reward Process (MRP). The rate of return at a time t can be defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \ . \tag{3.30}$$

More useful, however, is the value of a state, the expected return for that state. It describes how the stay in a state or the average transitions to a state is and can be calculated by averaging a large number of chains so that it is defined as follows:

$$V(s) = \mathbb{E}[G|S_t = s] \ . \tag{3.31}$$

When considering not only a passive agent that only observes a self-running system but an agent that acts in a system, the MRP must be extended to a Markov Decision Process (MDP) that contains actions that lie in the agent's action space. The transitions are then not only probability-based and the rewards do not only depend on the states, but both additionally depend on the chosen actions of the agent.

In most cases, an agent observes only a part of its environment, since it is very complicated or even impossible to obtain the complete state of the environment, as described in Section 3.4. In such a case, the MDP is extended to a Partially-Observable Markov Decision Process (POMDP).

The agent must somehow choose actions. This can be done, for example, with a strategy or with a set of rules that control the agent's behavior. The main goal of an agent in RL is to get as much out of it as possible, so it is important to implement a good strategy. Formally, it is defined as the probability distribution over actions for each state:

$$\pi(a|s) = P[A_t = a|S_t = s] \ . \tag{3.32}$$

However, this definition only tells how the actions of a state are distributed, but not directly how the actions should be selected. Therefore, different algorithms try to approximate a good policy to maximize the return [Lap20].

3.4.3 Taxonomy

RL methods can be divided into different aspects:

- Model-free or model-based
- Value-based or policy-based
- On-policy or off-policy

If a method is model-free, it means that it does not explicitly model the environment or reward, but simply builds a model from observations and actions. On the other hand, model-based methods try to predict observations or rewards with a model.

Policy-based methods directly approximate a policy, which is a probability distribution over the available actions. Value-based methods, on the other hand, compute the values of actions and select the action with the best value at each step.

Off-policy methods can train with historical data because only action values are updated. For this reason, very large experience buffers can be used for training to bring the data used for training closer to the iid property. Training data for on-policy methods, on the other hand, must always be sampled from the current policy because it depends on it and it is (in-)directly updated.

Off-policy methods tend to converge slower because of training with a large data history. On-policy methods cannot use historical data, but they tend to converge faster [Lap20].

3.4.4 Cross-Entropy method

The Cross-Entropy (CE) method is a simple policy-based method that has good convergence for simple environments. Here, a nonlinear function, usually a DNN, generates the policy,

which is a probability distribution over actions for given observations. The actual action chosen is determined by a random sample from such a generated probability distribution returned as input for observations. The basic idea here is to train only on elite episodes whose total rewards are above a threshold.

The method has the limitation that the episodes must be finite and preferably short, and good episodes must be separable from bad ones within the samples, i.e., they must have sufficient variability. In addition, there is no intermediate indication of whether the agent was successful or not [Lap20].

The CE method can be used for estimation and optimization. In RL, it is used to optimize the DNN. Here, the optimization problem is first converted into a rare event estimation problem, and then the CE method is used as an adaptive algorithm to find local minima.

The importance sampling theorem,

$$\mathbb{E}_{x\sim p(x)}[H(x)] = \int_x p(x)H(x)dx = \int_x q(x)\frac{p(x)}{q(x)}H(x)dx = \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}H(x)\right] \ , \tag{3.33}$$

is thereby used to estimate the entropy $H(x)$ for a policy x sampled from the distribution of all possible policies, i.e., $x \sim p(x)$. Random or brute-force search for all possible policies to maximize reward is inefficient and therefore impractical. Instead, a good policy is iteratively approximated by minimizing the distance between the probability distribution $p(x)$ and the optimal distribution $p^*(x)$, according to the theorem, i.e., by approximating $q(x)$ to $p(x)H(x)$.

The distance between two probability distributions can be calculated with the Kullback-Leibler (KL)-divergence:

$$KL(p_1(x) \parallel p_2(x)) = \mathbb{E}_{x\sim p_1(x)} \log \frac{p_1(x)}{p_2(x)} = \mathbb{E}_{x\sim p_1(x)}\left[\log p_1(x)\right] - \mathbb{E}_{x\sim p_1(x)}\left[\log p_2(x)\right] \ . \tag{3.34}$$

Of these, only the second term, the negative log-likelihood (NLL), is relevant to optimization, since it depends only on $p_2(x)$. Any loss function consisting of an NLL is a CE between the empirical distribution through the training data set and the modeled probability distribution [GBC16].

Combining the NLL with the theorem leads to an iterative algorithm,

$$q_{i+1}(x) = \underset{q_{i+1}(x)}{\operatorname{argmin}} -\mathbb{E}_{x\sim q_i(x)}\frac{p(x)}{q_i(x)}H(x)\log q_{i+1}(x) \ , \tag{3.35}$$

which starts with $q_0(x) = p(x)$ and improves through each step.

In the RL, this can be simplified by replacing the entropy $H(x)$ with the indicator function. The strategy update then looks like this [Lap20; Kro]:

$$\pi_{i+1}(a|s) = \underset{\pi_{i+1}}{\operatorname{argmin}} -\mathbb{E}_{z\sim\pi_i(a|s)}\left[R(z) \geq \psi_i\right]\log \pi_{i+1}(a|s) \ . \tag{3.36}$$

3.4.5 Bellman equation of optimality

In Section 3.4.2 the value of states was introduced. In the deterministic case, all actions have fixed rewards. The value of a state, knowing the values of subsequent states, can be calculated by the maximum of the reward and discounted value of a subsequent state taken by an action:

$$V_0 = \max_{a \in A}(r_a + \gamma V_a) \ . \tag{3.37}$$

For the stochastic case, where actions perform probability-based state transitions, the calculation must be extended with the expected value for each action:

$$V_0(a) = \mathbb{E}_{s \sim S}\left[r_{s,a} + \gamma V_s\right] = \sum_{s \in S} p_{a,0 \to s}(r_{s,a} + \gamma V_s) \ . \tag{3.38}$$

Combining these two relationships, the Bellman equation of optimality for a general case can be derived as follows:

$$V_0 = \max_{a \in A} \mathbb{E}_{s \in S}\left[r_{s,a} + \gamma V_s\right] = \max_{a \in A} \sum_{s \in S} p_{a,0 \to s}(r_{s,a} + \gamma V_s) \ . \tag{3.39}$$

This definition is recursive since the value of a state is defined over the values of the next reachable states. The computation of this can be done by iteratively updating the values, converging to the optima. The values describe not only the best reward but also the optimal strategy since an agent with the values of the states knows how to obtain all these rewards [Lap20].

3.4.6 Q-values

Since the agent wants to get the highest rewards, it must get to the states with the highest values as efficiently as possible. It can do this by considering all the values from the nearest reachable states in a state at a time and choosing the highest one. However, this computation can be costly. Therefore, to simplify the agent's decision of which action to take, state action values are introduced as an adjunct to the values of the states, as described in Section 3.4.5. These Q-values are defined by:

$$Q(s,a) = \mathbb{E}_{s' \in S}\left[r(s,a) + \gamma V(s')\right] = \sum_{s' \in S} p_{a,s \to s'}(r_{s,a} + \gamma V(s')) \ , \tag{3.40}$$

thus:

$$V(s) = \max_{a \in A} Q(s,a) \ . \tag{3.41}$$

They can be iteratively caluclated by:

$$Q(s,a) = r(s,a) + \gamma \max_{a' \in A} Q(s',a') \ . \tag{3.42}$$

Q-values are more convenient for the agent in decision-making than state values alone since it can compute them directly and choose actions based on them. In the stochastic case, an agent must instead estimate probabilities for transitions, since these are rarely known in advance. The calculation of Q-values can be used algorithmically in the simplest form in the value iteration method. This method uses tables for rewards, transitions, and values to perform the update, which is not described further here [Lap20].

3.5 Deep Reinforcement Learning

The value iteration method, as mentioned in Section 3.4.6, needs to know all states in the environment because it wants to compute values for all states. However, in more complex and thus usually larger environments, there are too many states that are potentially not observed in the agent's exploration. Another problem is that the value iteration approach does not support continuous action spaces because the value approximations assume that the actions are mutually exclusive discrete sets. One solution to the problem of iterating over the entire set of states is to use a blending technique that extends value iteration to Q-learning. The Q-value updates are then performed as follows:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in A} Q(s',a')) \ . \tag{3.43}$$

The specific algorithm is not further described here, as it is the basis for further methods [Lap20].

3.5.1 Q-learning

In simple Q-learning, the problem of the unmanageable number of states that need to be identified and their values approximated in larger environments remains. As a solution to this problem, non-linear representations such as DNNs, as described in Section 3.3, can be used that map both the state and the action to a value. The model is then called Deep Q Network (DQN).

Tuples (s,a,r,s'), which are sampled by interacting with the environment, are used as training data for this model. They are then fed into the approximating policy model $Q(s,a)$, which is optimized using SGD, as described in Section 3.3.4. Here, if the episode is terminated, the loss function $\mathcal{L} = (Q(s,a) - r)^2$ is used, otherwise the loss function $\mathcal{L} = \left(Q(s,a) - \left(r + \gamma \max_{a' \in A} Q_{s',a'}\right)\right)^2$ is used. If the tuples are chosen randomly, then the agent only explores the strategy already learned but never exploits it, as described in the dilemma or tradeoff between exploration and exploitation in Section 3.5.5. Therefore, the

epsilon-greedy method is used as a solution, which switches between using random and Q-value strategies for interacting with the environment depending on the probability hyperparameter ϵ.

For the SGD optimization algorithm, the data must be iid. However, since the sampled tuples from the same episode are very close to each other they are not independent. They are also not uniformly distributed over the optimal strategy, but over the current or random strategy. To meet this requirement as best as possible, a replay buffer is used that always stores only the most recent experiences. This increases the independence of the data, as there are more episodes, but the training data is still recent enough.

Some extensions for DQN improve various things, but they are not covered here [Lap20].

3.5.2 Policy gradients

Q-learning, as described in Section 3.5.1, tries to choose a good action to take by indirectly approximating the value of states and taking the action to the best state. In RL, the values of states are not relevant, but the policy maps observations to actions. This can also be learned directly with *policy gradient methods*. In contrast to the discrete action spaces of the original DQN method, they can handle continuous action spaces. They also can map to action distributions, from which concrete actions are sampled, to incorporate stochasticity and smooth representations.

In Q-learning, as described in Section 3.5.1, an attempt is made to choose a good action by indirectly approximating the value of the states and applying the action that leads to the best next state. In RL, the values of the states are not relevant, but the strategy that maps observations to actions is. This can also be learned directly with *policy gradient methods*. Unlike the discrete action spaces of the original DQN method, they can handle continuous action spaces. However, they can also be mapped to action distributions from which specific actions are retrieved to account for stochasticity and smooth representations.

For this method, the political gradient is used, which is defined as follows:

$$\nabla J \approx \mathbb{E}\left[Q(s,a)\nabla \log \pi(a|s)\right] \ . \tag{3.44}$$

It specifies the direction in which to improve the parameters of the network, as described in Section 3.3.4.2 since the goal in it is to improve the strategy in terms of the cumulative total reward, as described in Section 3.4.2. Since the scale of the gradient is proportional to $Q(s,a)$, the probability of actions with high total reward is increased and with low total reward is decreased. It can be implemented with the loss function,

$$\mathcal{L} = -Q(s,a)\log \pi(a|s) \ . \tag{3.45}$$

used for DNN policy optimization as described in Section 3.3.4.2, as in the vanilla policy gradient, the REINFORCE algorithm.

The policy gradient method extends the CE method as described in Section 3.4.4 by refining the separation of episodes with arbitrary values for $Q(s, a)$, since the CE method uses only $Q(s, a) = 1$ for good episodes and $Q(s, a) = 0$ for bad episodes.

Policy gradient methods do not require explicit exploration, unlike Q-learning which uses the epsilon-greedy method as described in Section 3.5.1. No replay buffer is used either, so policy gradient methods belong to the class of on-policy methods. They, therefore, converge faster but also require much more interaction with the environment and are therefore less sampling efficient. In addition, no target network is needed as in DQN, since the Q-values used are obtained directly from the interaction and are no longer approximated.

Policy gradient methods can only train on complete episodes. Long episodes thus exacerbate the sampling efficiency problem. Therefore, the estimation of Q-values is performed as in Q-learning. Since policy gradient methods do not have explicit state values, they are first estimated by *Actor-Critic-methods*, from which the Q-values are then obtained.

Replay buffers cannot be used in policy gradient methods because the policy update is based on the last interaction samples. However, replay buffers were introduced in Q-learning to solve the problem of overly correlated samples. This problem is solved in policy gradient methods by using transitions from multiple, parallel environments.

Replay buffers cannot be used in policy gradient methods because policy updating is based on the last interaction samples. However, replay buffers were introduced in Q-learning to solve the problem of overly correlated samples. This problem can be solved in policy gradient methods by using transitions from multiple parallel environments. But the REINFORCE algorithm still performs poorly in more complicated environments due to training instabilities [Lap20].

3.5.3 Actor-critic

The stability of the vanilla policy gradient method can be improved by reducing the variance,

$$\mathrm{Var}[x] = \mathbb{E}[(x - \mathbb{E}[x])^2] \ , \tag{3.46}$$

of the gradient. One approach is to subtract the mean total reward, called the baseline, from the gradient scale, the Q-value.

Another approach to reducing variance is to make the baseline dependent on the state. For this, the total reward can be represented as the value of a state $V(s)$ plus the benefit of an action $A(s, a)$: $Q(s, a) = V(s) + A(s, a)$. When $V(s)$ is used as the baseline, the gradient is simply the better advantage $A(s, a)$. Since $V(s)$ is unknown, it can also be approximated by a DNN, the critic, in parallel with the policy network, the actor. The critic is only needed and used in the training phase; for inference, on the other hand, the actor network alone maps from observations to actions. This method is used in the A2C algorithm but is not described further here.

For continuous action spaces, the policy network can directly output actions. For better exploration, parameters can be returned for a probability distribution over the actions, from

which the actual actions can be sampled. In *deterministic policy gradient methods*, such as the algorithm Deep Deterministic Policy Gradient (DDPG), the policy network, $\mu(s)$, converts states to actions by directly deterministically mapping observations to actions. The critic deterministically estimates the value $Q(s,a) = Q(s,\mu(s))$ for states and actions. By applying the chain rule as described in Section 3.3.4.1, the gradient for updating can then be derived as follows:

$$\nabla_a Q(s,a) \nabla_{\theta_\mu} \mu(s) \ . \tag{3.47}$$

The DDPG algorithm has the advantage over the Advantage Actor-Critic (A2C) algorithm in that it can be optimized from end to end since the entire system is differentiable and can train from a replay buffer [Lap20].

3.5.4 Taxonomy

DRL algorithms can be classified according to the general taxonomy rules of Section 3.4.3, as shown in Figure 3.10.

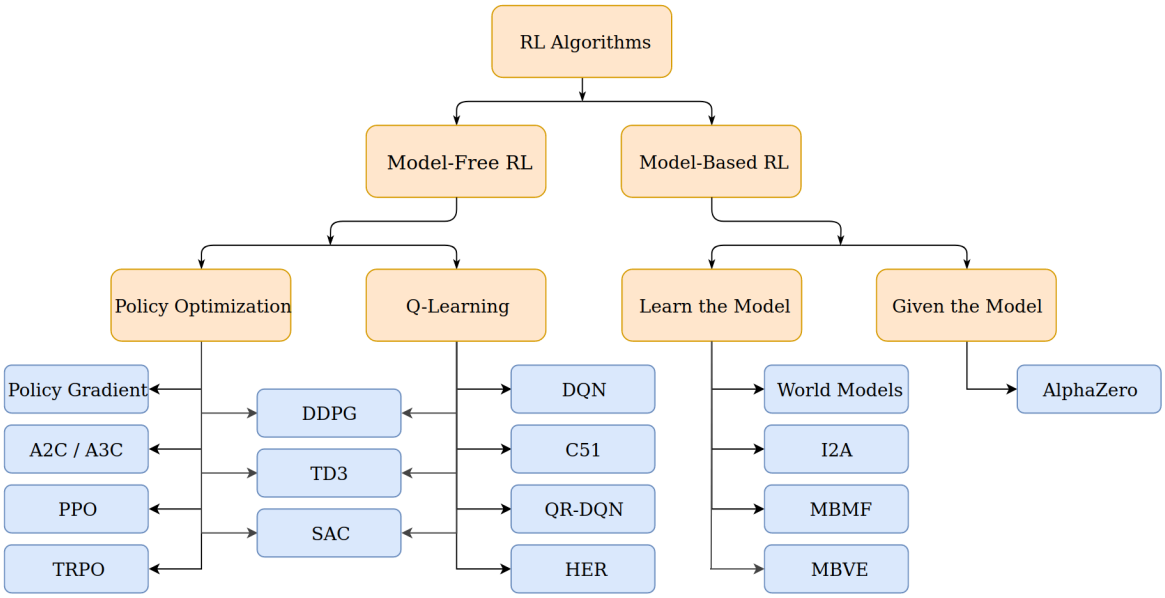


Figure 3.10: Taxonomy of DRL algorithms, from [Ope23]

All algorithms described here are model-free, i.e. they do not predict the next observations. Within this class, there are the classes of policy-based optimization and value-based Q-learning. Policy optimization methods, such as the policy gradients and A2C described above, train directly on the policy. Q-learning algorithms, such as. DQN, train outside the policy with a replay buffer. DDPG combines these two classes by training on a policy- and value-based network with a replay buffer [Ope23].

3.5.5 Autocurriculum learning

In multi-agent intelligence research, there is a hypothesis that MAS can generate momentum (intrinsic dynamics) through competition and cooperation. But MAS are not limited to this, nor do they imply it. According to the hypothesis, it promotes innovation and is modeled after evolution, where different entities also interact and adapt to each other. According to this hypothesis, innovation occurs when parts of the system are pushed by others into unknown search space regions where previously applied solutions no longer work, and therefore new ones must be developed. As in evolution, challenges become more complex over time, forcing actors to develop more and more innovations [Lei+19].

Challenges could be built in by adapting the environment, but this becomes quite complicated over time as the complexity of the environment also increases. Thus, the underlying environmental dynamics are dynamically changed by other agents. When an agent exploits its policy, the environmental dynamics are changed by its actions and thus the observed states of other agents. These other agents must then further explore this search space of the environment in which all agents are located. They then learn with Exploration by Exploitation by following the gradient of their experience, rather than the traditional Exploration vs. Exploitation as described in Section 3.4.1.

In general, a sequence of such challenges is called a curriculum. If the challenges are generated by the Exploration by Exploitation paradigm, they are referred to as an *autocurriculum* because they arise naturally from the non-stationary dynamics of the agents' social interaction processes. There is then no need for manual environmental engineering, as the challenges are generated by the system itself [Lei+19] [Bak+20].

3.5.6 Exogenous

Challenges can be divided into exogenous and endogenous challenges. An exogenous challenge originates outside the adaptive unit under consideration. When an agent changes its strategy, this leads to a change in the best response strategy of another agent. This other agent must then adjust its best response strategy. All agents improve their strategies, which leads to other agents having to adapt their strategies as well. This can lead to a long sequence of novel challenges [DK79], which can challenge the capabilities of all agents in an exogenous autocurriculum.

Challenges can be divided into exogenous and endogenous challenges. An exogenous challenge originates outside the adaptive unit under consideration. When an agent changes its strategy, this leads to a change in the response strategy of another agent. This other agent must then adapt its best response strategy. As each agent improves its strategy, other agents must also adjust their strategies accordingly. This can lead to a long sequence of novel challenges [DK79] that challenge the capabilities of all agents in an exogenous autocurriculum.

This sequence of challenges may contain new challenges each time, but this is not guaranteed. This is the case, e.g., when the dynamics of the environment are oscillating, resulting in constantly recurring challenges. One idea to solve this problem is to continuously train an adaptive entity to defeat previous versions of itself. In such a self-play, an adaptive unit

competes against itself, which leads it to learn to exploit its own mistakes. In this way, it challenges itself to correct these mistakes the next time it is confronted with them [Lei+19].

While this sequence of challenges may contain new challenges each time, e.g., when the dynamics of the environment oscillate, resulting in constantly recurring challenges, this is not guaranteed. One idea to solve this problem is to continuously train an adaptive entity to defeat previous versions of itself. In such a self-game, an adaptive entity competes against itself, which leads to learning to exploit its flaws. When an agent is confronted with them several times, it challenges itself to correct those mistakes.

In the self-play, catastrophic forgetting must also be prevented, as newer generations may not be able to defeat older generations [Sam+13]. As a solution, an adaptive unit can play in self-play not only against the newest and thus strongest policy but against a large and as diverse set of older policies as possible to implement strategies against them as well. However, there are also cases where a certain kind of forgetting is not always harmful, e.g., when old, bad strategies are discarded later in training in favor of new, better variants [Lan+17].

3.5.7 Endogenous

At an atomic level of adaptive units, only exogenous challenges are possible. At higher hierarchical levels, adaptation can also be driven by endogenous challenges that contribute to the formation of collective integrity. In this case, competition between subunits of adaptive components is largely suppressed. This creates tension between individual and collective rationality [Rap74].

The suppression may break down. Then individuals take advantage of a shared resource [Ost90], reducing the remaining available quantity at least somewhat. When the total amount exceeds its share of the cost of continued exploitation, the resource is eventually taken. This is called the tragedy of the commons [Har68] [Ost90]. Individuals cannot act unilaterally to escape this fate, as their influence would be too small to make a difference. When faced with challenges associated with a social dilemma, innovation at the individual level is not enough. Therefore, the solution must be to change the behavior of a critical part of the participants [Sch73], who act together to achieve a higher goal.

One way to do this is to incorporate the concept of an institution. This represents a system of rules, norms, or beliefs, structuring adaptation processes at the individual level to ensure that the group as a whole achieves a socially beneficial outcome in the form of innovation at the collective level. An example of this is that a group can sanction over-exploitation through individual strategies so that this strategy is no longer dominant [Hug+18; Lei+17; Per+17]. The maintenance of an institution itself also depends on the interactions among participants. This can lead to a *second-order social dilemma*, also known as the *second-order free rider problem*, since each individual would prefer that others bear a greater share of this burden [Axe86; Hec89; Yam88].

Such dilemmas can be resolved by institutions formed at higher levels. These higher-order institutions in turn create successor dilemmas at even higher levels [Ost00]. One could thereby speak of such social dilemmas having an NFL property, as described in Section 3.2.1. Once one social dilemma is solved in one place, another emerges in another place. This dynamic

can produce a series of endogenous challenges that steadily increase in scope and complexity, forming an endogenous autocurriculum [Lei+19].

3.6 Explainability in Deep Reinforcement Learning

DNNs, as described in Section 3.3.2, represent black boxes because it is difficult to explain each of their parameters, so the important features and internal handling are not directly understandable. Therefore, conventional DRL methods also become black boxes [JVW20; ZBM16] because they contain DNNs, as in Section 3.5, so experts cannot directly understand how the agent understands the environment or why it chooses a particular action. This intransparency therefore limits trust in DRL agents [Qin+22].

eXplainable Artificial Intelligence (XAI) aims to explain DNN models by providing details or reasons to make their operation clear or easy to understand for a particular audience [Arr+20]. For example, they use feature relevance techniques. XRL is a subset of XAI intended for general public applications. This calls for ethical, responsible, and trustworthy algorithms for different audiences. Explainability is especially important in critical environments, where agent behavior must be rigorously justified with precision.

In the literature, there is no officially agreed definition of XRL and thus meaning of explainability and interpretability in this context, as there are different understandings of these terms. Interpretability is understood, for example, as a person's understanding of model decisions or the degree of consistency in predicting model results. Explainability is seen, for example, as a post-hoc property or a surface representation of interpretations. From this, one could summarize that interpretability aims at the overall structure of a model at a higher level, while explainability aims more concretely at the processing of a model. However, here it is used synonymously [Qin+22].

3.6.1 Readability vs performance

In general, XAI has a tradeoff between model readability and performance. Tree-based models are, for example, more readable, i.e., more transparent, than DNNs, as described in Section 3.3.2, but their performance is worse. Not only performance but also explainability is crucial for the use of a system. If a system is not trustworthy, especially in critical environments, it will not be used. In the case of DRL, as described in Section 3.5, there may be concerns about their correctness, or at least doubts that the black box system in question does not always behave as it should to achieve a particular goal [PV20b].

3.6.2 Scope and time

There are different types of interpretability in terms of the scope and timing of information acquisition. On a coarse level, models are either globally or locally interpretable and either intrinsically or post-hoc interpretable. Here, scope refers to the explained domain of the model in question and the time of information acquisition [PV20b].

In Figure 3.11, a taxonomy for mapping XAI methods to post-hoc and intrinsic (transparent algorithms) method classes is shown.

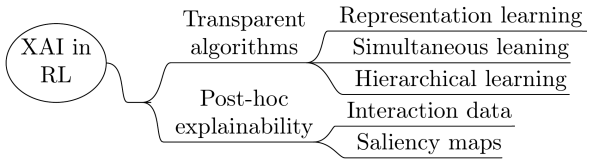


Figure 3.11: Conceptual model of the explanation process in XAI, by [Hof+19]

3.6.2.1 Transparent algorithms

An intrinsic model is directly interpretable on its own, without the need for additional processing at the time of its creation, as with a DT, linear regression, or rule-based system. However, conventional DRL algorithms are inherently intransparent because the DNN models used are black boxes [PV20b; HCD21].

Representation learning involves learning abstract features with low dimensionality. When learned from states, actions, or policies, this method can be used to explain RL models. State Representation Learning (SRL) [Les+18] aims, for example, to represent the state space. It can help to understand which features are relevant to learn to act on. Other representation learning methods include learning disaggregated representations, combining symbolic AI with DRL, inductive logic programming with self-attention, and the subdivision of world state representations into sub-states.

The standard DRL methods can also be optimized to learn policy and explanations simultaneously. These methods are recommended when external knowledge can be added to them. One method is to decompose rewards into a sum of meaningful reward types. It aims at explaining what kind of reward an agent wanted to maximize by an action. Explaining reward differences allows for understanding why one action has an advantage, i.e., why it is preferred over another. Models of action influence, such as structural causal models, learn causal relationships and thus explain why actions in certain states are or are not preferred to others.

Explanations can also be obtained by hierarchical goals. In hierarchical RL and sub-task decomposition, the main goal of high-level agents is decomposed into sub-goals for low-level agents, which in turn are used to perform high-level tasks. In this process, the representation of high-level agents is more interpretable by humans [HCD21].

3.6.2.2 Post-hoc explainability

Post-hoc interpretable models are created only after learning, e.g., by transforming or distilling a black-box DNN model into a reasonably interpretable model [PV20b; HCD21]. Examples of post-hoc XAI methods for standard ML models include SHapley Additive exPlanations (SHAP) [LL17] and the universal but only locally applicable LIME [RSG16].

When RL algorithms work with visual input, explanations can be obtained through saliency maps. They highlight the salient elements, e.g., the regions with the most relevant information.

Agent behavior can be explained in a generic way using interaction data with the environment. Key information or elements of interest can be extracted from the analysis of this data [HCD21].

3.6.3 Audience

XAI may include different goals, such as trustworthiness, causality, transferability, informativeness, fairness, confidence, accessibility, interactivity, and privacy awareness. The goals may differ, and so may the expected nature of the explanation, because it depends on the goal being pursued. Thus, the explanation depends on the target audience, as they may have different goals. The comprehensibility of the explanation depends on the transparency of the model for the explanation used, but also on the human interpretation of the explanation [HCD21].

There are different classes of audiences, [Arr+20] according to [HCD21], such as experts, users, developers, managers, and regulators. The experts, for example, have domain knowledge and are model users. The pursued goal of explainability methods here is to increase the trustworthiness of the model for the experts [HCD21].

3.6.4 Explanation evaluation

Evaluating the XAI method has proven to be a difficult task, as there is no single accepted concept and definition, and thus no clear consensus on which metric to use for evaluation [HCD21; Qin+22]. [DK17] proposes to use the categories of application, human, and function for XAI evaluation methods.

The evaluation of XAI methods has proven to be a difficult task, as there is no single accepted concept and definition, and thus no clear consensus on which metric to use for evaluation [HCD21; Qin+22]. According to [HCD21; Qin+22], the evaluation methods of XAI should be categorized as application, human, and function for evaluation methods of XAI.

XAI evaluation is complicated by audience dependence, as described in Section 3.6.3. Explanations are therefore qualitative and subjective, as well as subject to human variability. These problems are addressed by attempting to reconstruct the audience's mental model by conducting interviews with quantitative metrics such as accuracy and clarity of explanations to test the recipient's understanding. In [Hof+19], an evaluation method for such mental models is thereby proposed. In this context, mental models are interpretations of internal representations, i.e., mechanisms of the real world based on experience, which can be mentally simulated. In Figure 3.12, the XAI explanation process is presented for this purpose, focusing on the audience.

Initially, a user may feel that a model under consideration is not trustworthy. Explanations are then provided by XAI methods that build a mental model. Further explanations can refine the user's mental model, which should lead to better performance, reasonable confidence, and reliability [HCD21].

In addition to the evaluation of the mental model, there are also evaluations of the user-oriented properties, both of which belong to the subjective evaluations. Complementary to

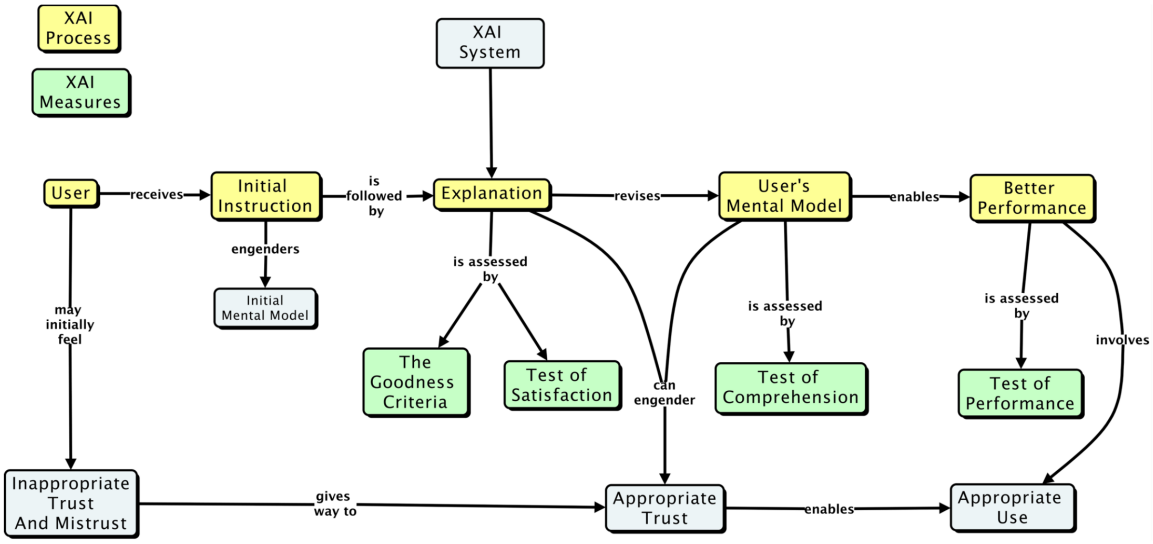


Figure 3.12: Conceptual model of the explanation process in XAI, by [Hof+19]

these, there are also objective evaluations, such as model performance, explanatory accuracy, and sensitivity and robustness, but these are not described further here [Qin+22].

3.7 Satisfiability Modulo Theories

CPSs are found in various application areas, such as software and hardware verification, type inference, static program analysis, test case generation, scheduling, and planning and graph problems where logical formulas are used to describe states and transformations between them.

The decision problem of whether a formula formed with logical connectives over Boolean variables can be satisfied by choosing true or false values for its variables is called the propositional SATisfiability (SAT) problem, which is a well-known Constraint-satisfaction problem (CSP). When these problems involve richer languages, such as arithmetic, supporting theories are required to capture the meaning of the formulas. Solvers for combining SAT problems and supporting theories are called SMT solvers, such a solver is, for example, Z3 [DB08].

SMT solvers use the SMT-LIB exchange format for benchmarks. They have been used in the context of program verification and advanced static verification, where verification focuses on assertion checking, interactive theorem proving, scheduling, planning, and software development.

In job scheduling, the decision problem is which of the jobs specified with the start time $(t_{i,j})$ should be executed one after another, where two jobs requiring the same machine cannot be executed simultaneously, but the duration of each job $d_{i,j}$ must not be greater than the specified maximum total time max. In addition, the tasks must not be interrupted and the

start time of the first task of each task must be greater than or equal to zero. These conditions can be encoded using the following SMT formulas, but they are not explained further here:

$$t_{i,j+1} \geq t_{i,j} + d_{i,j} \tag{3.48}$$

$$\underbrace{\underbrace{(t_{i,j} \geq t_{i',j} + d_{i',j})}_{\textit{atomic formula, also literal}} \quad \vee \quad (t_{i',j} \geq t_{i,j} + d_{i,j})}_{\text{conjunction of literals}} \tag{3.49}$$

$$t_{i,1} \geq 0 \tag{3.50}$$

$$t_{i,m} + d_{i,m} \leq \max \tag{3.51}$$

$$. \tag{3.52}$$

SMT-solvers decide the satisfiability of conjunctions of literals in the theory solution. A literal here is an atomic formula or its negation. These solvers handle subformulas as described in Eq. (3.49) by performing a case analysis with mostly efficient satisfiability methods for proportional logic, i.e., SAT problem solvers [DB11].

3.7.1 SAT problem solving

SAT problem solvers decide propositional logic, which is a special case of predicate logic. Such formulas, as shown in Eq. (3.49), are formed from Boolean variables and composed of logical connectives. Deciding SAT problems is NP-complete [Coo23], i.e., in general theoretically computationally intractable, but practically efficiently solvable by a systematic search approach. In this approach, the search space is modeled as a tree containing the variables and the choices.

SAT solvers are usually based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [DLL62] for this purpose. It restricts formulas to Conjunctive Normal Form (CNF), which may only contain conjunctions of clauses with disjunctions of literals. An example of a formula in this case is $\neg p \wedge (p \vee q)$. The algorithm efficiently searches the tree for configurations that satisfy the formula from which the tree was formed, using various techniques. If it can be deduced that no such configuration exists, the formula is not satisfiable [DB11].

3.7.2 Difference arithmetic solving

By combining an SAT solver, as described in Section 3.7.1, with a theory solver for difference arithmetic, the work scheduling decision problem described in Section 3.7 can be solved. Difference arithmetic is a subset of linear arithmetic in which the predicates are restricted to the form $t - s \leq c$ with variables t, s and numerical constants c. Atoms, such as $t_{i,j+1} \leq t_{i,j} + d_{i,j}$ with fixed $d_{i,j}$ from Eq. (3.48), can be put into this form with $t_{i,j} - t_{i,j+1} \leq -d_{i,j}$. For atoms such as $s \leq c$ and $s \geq c$, a special new variable, the null variable, z is introduced so that they satisfy the required form by $s - z \leq c$ and $z - s \leq -c$.

The formulas in this required form are used to form a weighted graph with nodes s, t and edges from s to t with weights c, i.e., $s \xrightarrow{c} t$. The satisfiability of the set of difference arithmetic atoms can then be efficiently checked by searching the graph for cycles with the negative sum

of the weights of its transitions. As an example, consider the difference arithmetic formulas on the left side of Figure 3.13, which belong to a set of work scheduling formulas. From them, the graph on the right side of Figure 3.13 was created.

$$
\begin{aligned}
z - t_{1,1} &\leq 0 \\
z - t_{2,1} &\leq 0 \\
z - t_{3,1} &\leq 0 \\
t_{3,2} - z &\leq 5 \\
t_{3,1} - t_{3,2} &\leq -2 \\
t_{2,1} - t_{3,1} &\leq -3 \\
t_{1,1} - t_{2,1} &\leq -2
\end{aligned}
$$

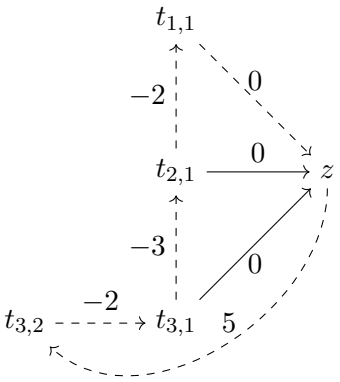


Figure 3.13: Example formulas and graph for difference arithmetic, from [DB11]

Since the graph contains a cycle (dashed) with a negative sum of weights of $-2-3-2+5 = -2$, the formulas are satisfiable. Thus, the following valid schedule can be derived for the example formulas: $t_{1,1} \rightarrow t_{2,1} \rightarrow t_{3,1} \rightarrow t_{3,2}$ [DB11].

3.7.3 Combination of SAT and difference arithmetic solving

The SAT and difference arithmetic solvers of Section 3.7.1 and Section 3.7.2 can be combined, as mentioned earlier in Section 3.7.2. This is done by mapping the atoms of a SMT formula to free Boolean variables so that the new abstract formula can be processed by a SAT solver. If the SAT solver determines that the abstract formula is not satisfiable, then the SMT formula is also not satisfiable.

On the other hand, if it finds a configuration for the variables of the formula such that it is satisfiable, the difference arithmetic solver checks the set of literals found that are implied by the configuration of the variables. For example, $p_1 \rightarrow \mathit{false}, p_2 \rightarrow$ true is a model for the formula $\neg p_1 \wedge (p_1 \vee p_2)$ that induces the set of literals $\{\neg(a \geq 3), a \geq 5\}$. These two literals are not satisfiable together in the theory of arithmetic because there is no a for which both literals hold. This implies that the negation of the abstract formula, $p_1 \vee \neg p_2$, must be true. The original formula is then checked again in conjunction with this formula by the solver SAT, finding that the formula as a whole is not satisfiable. This can be repeated in a loop, but the process always converges since the set of atoms is finite, and thus only a finite number of theory lemmas are generated [DB11].

4 Related Work

4.1 More fine-grained XRL taxonomy

When considering the categories of readability vs. performance, as described in Section 3.6.1, and in particular scope and time, as described in Section 3.6.2, further categories can be derived. Qing et al. [Qin+22] therefore present a new proposal for a taxonomy of XRL methods, as shown in Figure 4.1, based on the categories of model, reward, state, and task explanation, as shown in Figure 4.2. It is also discussed how human knowledge can be integrated for improvements.

From the categories of readability vs. performance, as described in Section 3.6.1, and in particular scope and time, as described in Section 3.6.2, additional categories can be derived. For example, Qing et al. [Qin+22] present a proposal for a finer taxonomy of XRL methods, as presented in Figure 4.1, based on the categories of model, reward, state, and task explanation, as presented in Figure 4.2. It also discusses how human knowledge can be integrated for improvements.

The categories are described as follows:

- Model explainability here means that models have comprehensible logical operations in their internal structures that are trained by agents to be explainable.
- Explainability of rewards means that the reward function is reconstructed by an explainable model, which should improve understanding of how an objective affects an agent.
- State explainability refers to quantifying the influence of state characteristics on decision-making by adding attention-based submodules to the model.
- Task explainability aims to achieve an architectural level of explainability in complex environments through multi-level agents.

Methods for explaining models are divided into self-explainable and explanation-generating methods. The class of self-explainable models is related to inherently interpretable models as described in Section 3.6.2.1 but also includes methods that represent the policy model using self-explainable models. Explanation-generating models are not inherently explainable, but use additional explicit explainability logic to automatically generate explanations during training. This logic is learned from human understanding of tasks or ways of thinking to understand new things [Qin+22].

In the following, an overview of the relevant subcategories of model-based methods is given. The other categories of explainability of rewards, states, and tasks are not further described here.

4.1.1 Explanation-generating model-based methods

Explanations can be generated by counterfacts, human explanations, instructions, queries, or verification. Counterfactual explanations can help understand why an action or state was

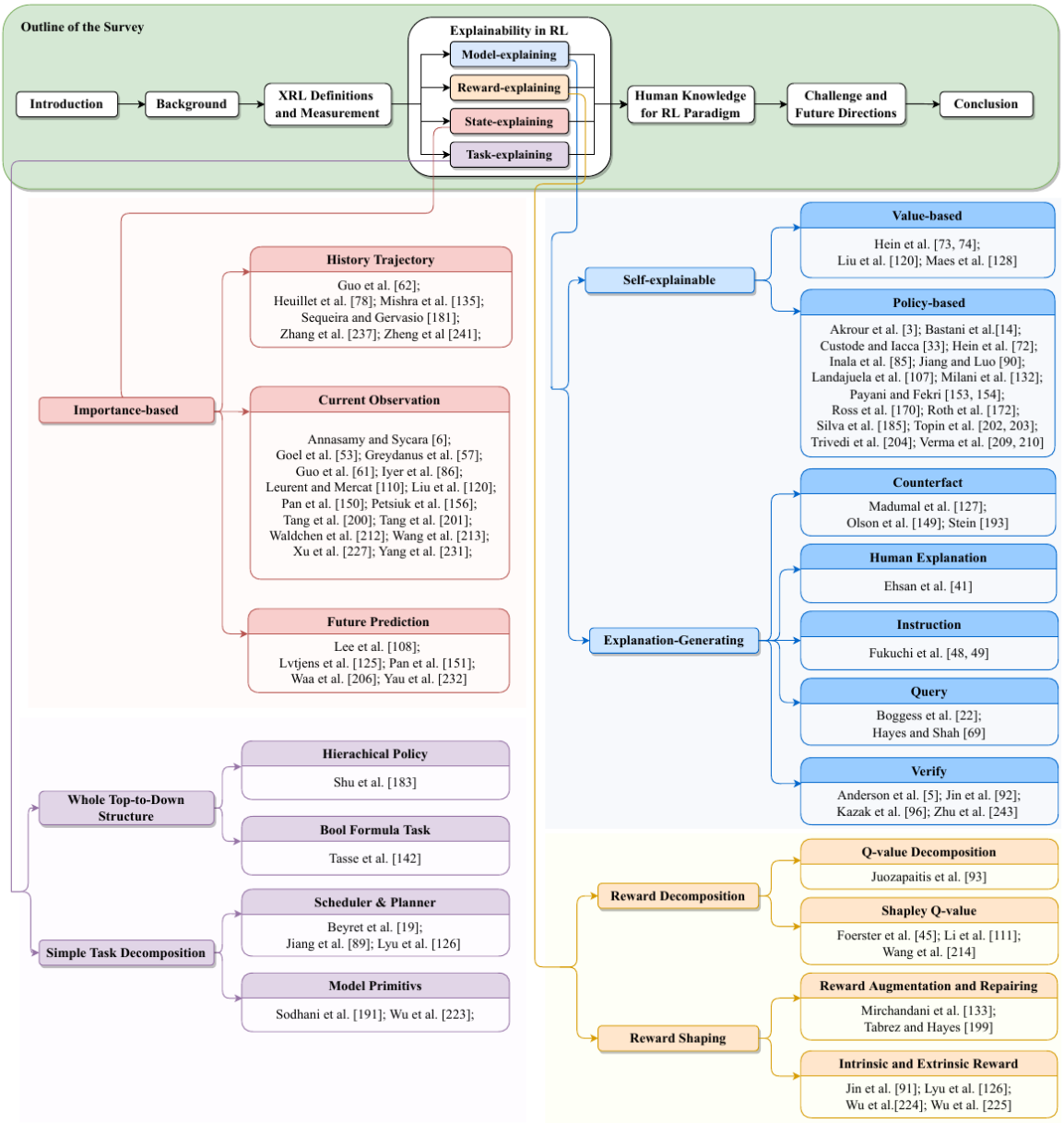
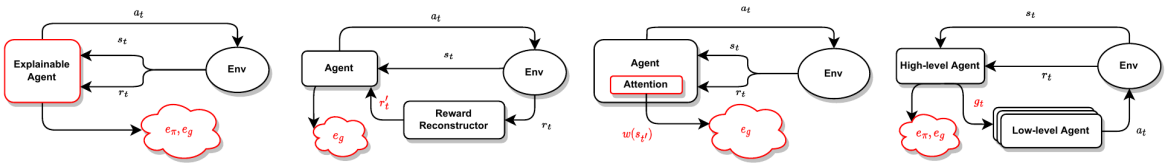


Figure 4.1: An overview of the [Qin+22] survey with presentation of the XRL taxonomy regarding classification by the RL parts: Model, Reward, Condition, and Task. Representative works are assigned to the categories.



(a) Model-explaining (b) Reward-explaining (c) State-explaining (d) Task-explaining

Figure 4.2: Abstract diagrams of the explanatory processes for the different XRL method categories, from and by [Qin+22]

chosen by explaining why the counterfactual action or state was not chosen in the simulation. Some methods generate explanations by learning the implicit explainability logic from human explanations in natural language. In instruction-based behavioral explanation, agents can learn instructions from human experts fed into an interactive RL, from which they can map the action goal to understandable expressions. Humans build a mental model of an environment by querying. This method can also be used in XRL by preprocessing human queries to find related states. These are then used to build a natural language summary. Formal verification techniques can also be used to explain agent models indirectly over the entire possible state space through counterexamples when logical verification fails [Qin+22].

Explanations can be generated by counterfactual explanations, human explanations, instructions, queries, or verification. Counterfactual explanations can help understand why an action or state was chosen by explaining why counterfactual actions or states were not chosen in the simulation. Some methods that generate explanations by learning the implicit explanatory logic from human explanations in natural language. Furthermore, there is instruction-based behavior explanation, where agents learn instructions from human experts. In this case, the instructions are fed into interactive RL that maps the action goal to understandable expressions. Another method is based on the concept that humans build a mental model of an environment through queries. In this, human queries are preprocessed to find related states. These are then used to build a natural language summary. Finally, formal verification techniques can also be used to indirectly explain agent models over the entire possible state space using counterexamples when logical verifications fail [Qin+22].

4.1.2 Self-explainable policy-based methods

Self-explainable methods can be further divided into value-based and policy-based methods, as shown in Figure 4.3, but only policy-based methods are described further.

Programmatic RL (PRL) represents the policy model of an agent using a program in the form of Domain Specific Languages (DSLs) whose logical rules can provide global explainability. Thereby, interpretable programmatic policies are generated in it by using program synthesis [FCD15]. It reconstructs the policy using generalizations of input-output examples from history, guaranteeing that it fits all patterns provided. Such methods can output source code that can be verified using traditional symbolic program verification techniques [Kin76]. In PRL, there are several methods for finding a programmatic policy with maximum reward by updating an already synthesized programmatic policy or by imitation learning. The formula

Type	Description	Model	Algorithm	Environments
Value-based	Learning while representing Q-value in an understandable way using different models	Decision Tree	Linear model U-tree [120]	Flappy bird, MountainCar, Cartpole
		Formula Expression	Depth-limited search [128]	Multi-armed bandits
			Genetic programming [73, 74]	MountainCar, Cartpole, Industrial control benchmark
Policy-based	Learning while representing policy in an understandable way using different models	Programmatic Policy	Programatic interpretable RL [210]	TORCS car racing
			Imitation-projected programmatic RL [209]	TORCS car racing
			Neurosymbolic transformers [85]	Formation task, Unlabeled goals task
			Learn program embedding space [204]	Stair climber, Four corner, Top off, Maze, Clean house, Harvester
		Symbolic Policy	Deep symbolic policy generator [107]	MountainCar, Pendulum, Hopper, InvDoublePend, InvPendSwingup, LunarLander, BipedalWalker
		Fuzzy Controller	Policy gradient [3]	Hopper, BipedalWalker
			Fuzzy particle swarm RL [72]	MountainCar, Cartpole
		Logic Rule	Neural logic RL [90]	Block manipulation, Cliff-walking
			Genetic programming RL [153, 154]	BoxWorld, GridWorld
		Decision Tree	Verifiability via iterative policy extraction [14]	Cartpole, Pong
			Muti-agent verifiability via iterative policy extraction [132]	Cooperative navigation, Predator-prey, Physical deception
			Iterative bounding MDP [202]	CartPole, PrereqWorld, PotholeWorld
			Abstraction policy graph [203]	PrereqWorld
			Q-value pruning [170]	Super tux kar, Super mario bros
			Evolutionary learning [33]	Cartpole, MountainCar, Lunarlander
			Conservatice Q-improvement [172]	Robot navigation

Figure 4.3: Self-explainable XRL methods by their class and model type with representative methods, from [Qin+22].

expression can directly represent a policy model by a symbolic policy by fitting a dataset to the symbolic state space.

Fuzzy controllers can represent the policy by fuzzy rules consisting of conditions and actions learned from DRL models. As another method, logical rules are used to represent the policy since they are much more understandable to humans. These are also trained with the history of condition-action interaction data.

DTs are tractable models that allow formal verification of policy behavior [BPS18], so they can be used to represent policy models. However, in classical RL, they find little application as approximation functions because they are poorly regularized compared to DNNs [BC14], but in XRL they are used for value- and policy-based methods. Here, training is used to generate features that can be explained to humans to understand entire tasks. One possible method to generate them is learning from a DNN-based policy model by imitation learning or distillation by resampling. Another approach to extracting a policy as a DT from a DNN-based policy model is to transform its rules.

The IBMDP method transforms DNN rules into a DT using an extended MDP as described in Section 3.4.2, the Iterative Bounding MDP described by Topin et al. [Top+21]. This allows the general CUSTARD procedure, applied as a modification of the standard DRL techniques, together with the standard DRL training of policy models, to extract equivalent DTPs of entire policy models. Here, a DTP is a DT that maps states to actions rather than observations. The DT is extracted in this method from an IBMDP policy by fitting edge observations from the environment [Qin+22].

4.2 DNN-DT equivalence description

In [NKA21], the EC-DT algorithm is presented that can transform FF-DNNs directly into DTs. An overview of this algorithm is therefore described in Section 4.2.1. Such a direct transformation algorithm is extended from Aytekin [Ayt22] to other DNN components, which is described in more detail in Section 4.2.2.

4.2.1 EC-DT algorithm

The EC-DT algorithm [NKA21] consists of two sequential algorithms, one for tree generation and one for rule extraction. The tree generation algorithm creates a binary DT from FF-DNN by annotating nodes from the iteration of linear layer weights and biases, and branching from and with activations. An example of the XOR function can be found in Figure 4.4.

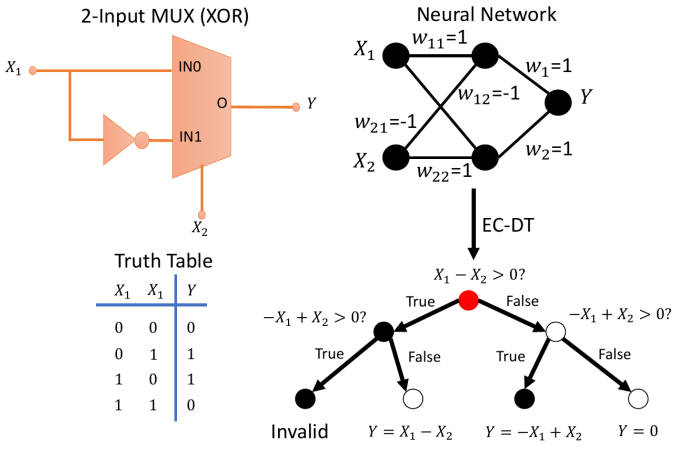


Figure 4.4: An example of an XOR gate with binary inputs that can be represented by an DNN and converted to an DT by the EC-DT algorithm, from [NKA21]

After the complete tree is generated, the algorithm can extract rules from the leaf nodes, which are converted into constraints using the DT components of the weight and bias matrices and the activations. A rule is a conjunction of activated constraints along a path. The constraints of a node may need to be changed according to the activation along the path. The following examples are not explicitly given in the paper, but are interpretations based on the EC-DT example in Figure 4.4, for potentially better understanding. The node constraint $-X_1 + X_2 > 0$ of the first node in the second row must be negated, i.e. converted to $-X_1 + X_2 \leq 0$, for the false branch. Thus, for the second path of the EC-DT, the complete rule is $X_1 - X2 > 0 \wedge -X_1 + X_2 \leq 0$.

Rules may contain conflicting conditions, as described in Section 6.3, which invalidate the rules. For example, the conditions of the first rule $X_1 - X_2 > 0$ and $-X_1 + X_2 > 0$ cannot be satisfied together and thus contradict each other, so the rule consisting of them is invalid. Invalid rules are subsequently eliminated to reduce the size of the rule set. The paper also does not explicitly state how the rules are verified, but the conjunction of constraints can be

checked for satisfiability using SMT solvers, as described in Section 3.7. Since this algorithm itself is not directly used further, it is not described further or in detail.

4.2.2 Equivalence description

Unlike the EC-DT algorithm described in Section 4.2.1, the equivalence description describes transformation rules from FF-DNNs to DTs [Ayt22]. Rules are given for transforming the linear layer and ReLU activation function, but also for CNN, RNN, and normalization layers, as well as for skip connection and continuous activation functions. Two algorithms are used to generate FF-DNNs, one of which is used as a method in the other. Thus, this is a combined algorithm instead of including two consecutive algorithms, as in the EC-DT algorithm.

These direct transform algorithms are formally derived from the definition of FF-DNNs. In Eq. (4.1) they are therefore expressed in closed form.

$$\begin{aligned} \mathrm{NN}(x_0) &= \boldsymbol{W}_{n-1}^\top \sigma(\boldsymbol{W}_{n-2}^\top \sigma(\dots \boldsymbol{W}_1^\top \sigma(\boldsymbol{W}_0^\top \boldsymbol{x}_0))) \\ x_i &= \sigma(\boldsymbol{W}_{n-1}^\top \sigma(\dots \boldsymbol{W}_1^\top \sigma(\boldsymbol{W}_0^\top \boldsymbol{x}_0))) \end{aligned} \tag{4.1}$$

Here $\boldsymbol{W}_i$ are the weight matrices of the i-th layer, σ is a piecewise activation function, and $\boldsymbol{x}_0$ is the input to the FF-DNN. The activation function σ is applied element-wise, so can be written as by element-wise scalar multiplication with the activation vector $\boldsymbol{a}$:

$$\boldsymbol{W}_i^\top \sigma(\boldsymbol{W}_{i-1}^\top \boldsymbol{x}_{i-1}) = \boldsymbol{W}_i^\top (\boldsymbol{a}_{i-1} \odot (\boldsymbol{W}_{i-1}^\top \boldsymbol{x}_{i-1})) \tag{4.2}$$

$$= (\boldsymbol{W}_i \odot \boldsymbol{a}_{i-1})^\top \boldsymbol{W}_{i-1}^\top \boldsymbol{x}_{i-1} \ . \tag{4.3}$$

Here, the vector $\boldsymbol{a}$ gives the slopes of the activations in the linear regions where $\boldsymbol{W}_{i-1}^\top \boldsymbol{x}_{i-1}$ falls. In Eq. (4.1), the element-wise multiplication of $\odot$ is applied column-wise to $\boldsymbol{a}_{i-1}$. So $\boldsymbol{a}$ is repeated k times to match the size of $\boldsymbol{W}_i$, so $\boldsymbol{a} \coloneqq [(\boldsymbol{a}^\top)_{\times k}]$.

Substituting Eq. (4.3) into Eq. (4.1) yields:

$$\mathrm{NN}(x_0) = (\boldsymbol{W}_{n-1} \odot \boldsymbol{a}_{n-2})^\top (\boldsymbol{W}_{n-2} \odot \boldsymbol{a}_{n-3})^\top \dots (\boldsymbol{W}_1 \odot \boldsymbol{a}_0)^\top \boldsymbol{W}_0^\top \boldsymbol{x}_0 \ . \tag{4.4}$$

Eq. (4.4) can be iteratively calculated from right to left, so that the output can always be calculated by left multiplication $(\boldsymbol{W}_i \odot \boldsymbol{a}_{i-1})^\top$ with the previously already calculated effective weight matrix $\hat{\boldsymbol{W}}_i^\top$, so that:

$$\begin{aligned} \boldsymbol{c}_{i-1} \hat{\boldsymbol{W}}_i^\top &= (\boldsymbol{W}_i \odot \boldsymbol{a}_{i-1})^\top \dots (\boldsymbol{W}_1 \odot \boldsymbol{a}_0)^\top \boldsymbol{W}_i^\top \\ \boldsymbol{c}_{i-1} \hat{\boldsymbol{W}}_i^\top \boldsymbol{x}_0 &= \boldsymbol{W}_i^\top \boldsymbol{x}_i \ . \end{aligned} \tag{4.5}$$

This can also be expressed as an algorithm for the activation function ReLU, as shown in Algorithm 1.

Algorithm 1
Algorithm for computing effective weight matrices with left-handed linear transformation for the ReLU activation function, from [Ayt22]

$\hat{\boldsymbol{W}} = \boldsymbol{W}_0$
for $i = 0, \ldots, n-2$ **do**
  $\boldsymbol{a} = [\,]$
  for $j = 0, \ldots, m_i - 1$ **do**
    if $\hat{\boldsymbol{W}}_{ij}^{\top} \boldsymbol{x}_0 > 0$ **then**
      $\boldsymbol{a}$.append(1)
    else
      $\boldsymbol{a}$.append(0)
  $\hat{\boldsymbol{W}} = \hat{\boldsymbol{W}}(\boldsymbol{W}_{i+1} \odot \boldsymbol{a}$
return $(\hat{\boldsymbol{W}}^{\top} \boldsymbol{x}_0)$

In Eq. (4.5), the vector $\boldsymbol{c}_{i-1}$ is defined by the concatenation of all previous activation vectors, so that $\boldsymbol{c}_{i-1} \coloneqq \boldsymbol{a}_0 \parallel \boldsymbol{a}_1 \parallel \ldots \boldsymbol{a}_{i-1}$ holds. Since $\hat{\boldsymbol{W}}_i^{\top}$ depends on the activations based on the input, a different $\hat{\boldsymbol{W}}_i^{\top}$ can be calculated for different inputs. Thus, a DT can be formed by branching based on the input for different activation vectors to calculate the effective weight matrices.

DTs can then be constructed with Algorithm 2, using the algorithm in Algorithm 1 as a method therein. Since this is the algorithmic representation of Eq. (4.5), which is simply a reformulation of the definition of FF-DNNs, a DT constructed using Algorithm 2 directly and accurately represents an FF-DNN used as the basis for the construction.

Algorithm 2
Algorithm of transformation from FF-DNNs to DTs, from [Ayt22]

1: Initialize Tree: Set root.
2: Branch all leafs to k nodes, decision rule is first effective filter.
3: Branch all nodes to k more nodes, and repeat until all effective filters in a layer is covered.
4: Calculate effective matrix for each leaf via Eq. (4.5) Repeat 2,3.
5: Repeat 4 until all layers are covered.
6: **return** Tree

In a DT generated by Algorithm 2, an FF-DNN layer i is represented as a k^{m_i}-way categorization, where m_i is the number of filters in each layer and k is the total number of linear regions in an activation. For the ReLU activation function, where $\boldsymbol{a} \in \{0, 1\}$, it is a binary tree with $k = 2$. The DT then has a depth of $d = \sum_{i=0}^{n-2} m_i$ and a total number for the categories in the last branch of 2^d. Thus, DTs grow exponentially by construction, which is practically impossible to represent even for small FF-DNNs. However, the paper notes that violating and redundant rules can occur in such generated FF-DNNs, which allows lossless pruning. Another observation is that there may be categories (tree leaves) that are not realized by training with the provided training dataset, so they may also be considered invalid and thus pruned if data falls into them.

An example of a DT transected starting from an FF-DNN is shown in (b) of Figure 4.6. The FF-DNN learned to approximate $y = x^2$ in the process.

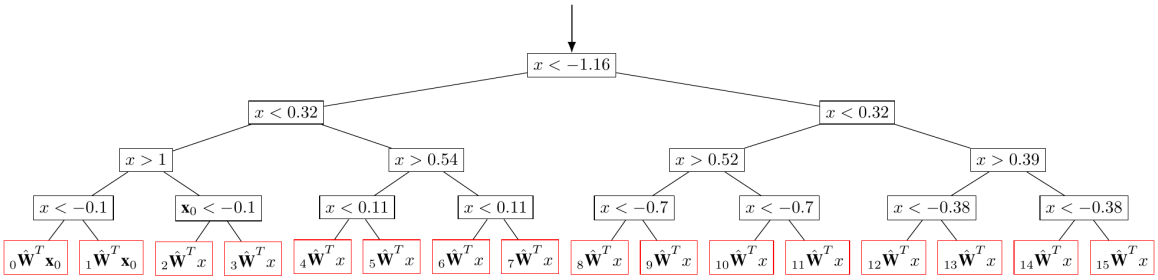
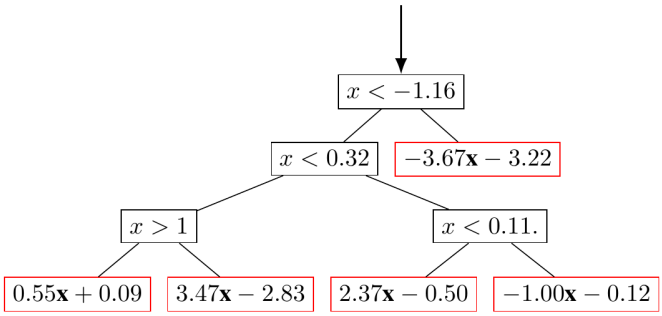
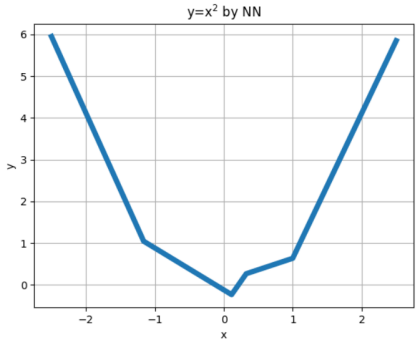


Figure 4.5: Transformed DT based on an FF-DNN approximating $y = x^2$, from [Ayt22]

In (a) of Figure 4.6, the cleaned DT based on Figure 4.5 is shown.



(a) Cleaned Tree



(b) Neural Network Approximation of $y = x^2$

Figure 4.6: The cleaned DT for the original transformed DT in Figure 4.5 in (a) and the representation of the approximated function $y = x^2$ of the FF-DNN, from [Ayt22]

5 Requirements, Method Analysis and Contributions

5.1 Requirements

In the following, requirements for an appropriate XRL method are derived that address the challenges described in the introduction in Section 2.2:

- (R1) The method MUST explain agent behavior in a way that experts can understand to verify that the model does not contain harmful behavior.
  - (R1.1) The method MUST directly explain the agent model, the policy model, as described in Section 3.5.2, and NOT indirectly through other components of agent inference, such as state, reward, or task, since such explanations target why or how the agent works, but may not be appropriate to let the domain expert verify that the agent does not exhibit harmful behavior.
- (R2) The method MUST explain the entire agent model (see (R1.1)), i.e., it MUST be global, as described in Section 3.6.2, to remove uncertainty about unknown unknowns in agent behavior.
  - (R2.1) The method MUST explain the agent model exactly, i.e., it MUST NOT rely on approximations of the agent model (see (R1.1)) since unknown unknowns cannot be eliminated when using approximations.
- (R3) The method MUST provide explanations for the agent's behavior at least post-hoc, but MAY also explain it during training since a model for an agent in a critical environment must be extensively tested and validated for harmless behavior before it can be used, as described in Section 2.2.
- (R4) The method MUST explain the agent model in such a way that verification methods can automatically apply it to verify the model itself, since a better explanation may require constraints from the pre-analysis of the explanations by domain experts in verification that they need to understand beforehand. Some explanations may also be too complicated, so the experts may want to check the properties of these explanations.
- (R5) The method MUST NOT be based on a modification of existing standard DRL methods OR be limited to a specific method, as it should be as universal as possible, to be used as a standard XRL method for as many DRL methods as possible in the future, to increase the knowledge and thus the confidence of domain experts in it. However, the method MAY be restricted to some classes of DRL methods, e.g., policy-based methods, otherwise (R1.1) could not be satisfied.

The requirements were created by analyzing the challenges in light of the fundamentals, so no claim is made to complete accuracy or completeness. Some requirements contain assumptions in their explanations. Whether these always apply in this form or whether others must be added must be verified by further requirements analysis based on literature research. If one of them does not apply, (partial) requirements can possibly be relaxed.

5.2 Method analysis

In this section, the classes of XRL methods are analyzed against the taxonomies from Section 3.6 and, in particular, the finer taxonomy from Section 4.1 against the requirements defined in Section 5.1. No claim of completeness is made for this analysis, nor is a dedicated literature review conducted.

As already stated in the assumptions of the requirements, only model-explaining methods come into question for the fulfillment of the requirements, since only they can directly explain the behavior of the agent. These are in turn divided into explanation-generating and self-explaining methods, whereby explanation-generating methods, as described in Section 4.1.1, usually only indirectly explain the agent's model. Moreover, they are often local or approximate further models and thus they do not satisfy the requirements (R1.1), (R2), and (R2.1). Only verification methods can generate explanations over the entire possible state space. Thus, exact statements can be made about whether assertions are true, e.g. whether certain sequences of states do not occur. However, these methods can only verify properties such as safety and liveness, but cannot explain how and why an agent behaves, causing (R1) not to be satisfied.

Since no method from the explanation-generating class satisfies all requirements, only the policy-based methods of the self-explaining models described in Section 4.1.2 remain to be analyzed, since the value-based methods do not satisfy (R1.1).

5.2.1 Self-explainable policy-based methods

PRLs methods seem to be promising because they can explain a policy model directly and globally, are guaranteed by their design to fit all provided samples, and the programs are understandable and allow verification, so they satisfy (R1), (R1.1), (R2), and (R4). They can be used during training and post-hoc, so they also satisfy (R3). However, the generated programs are generalizations and therefore may not represent exactly the same strategy learned by the agent and therefore do not satisfy the requirement (R2.1). Instead of policy models, the synthesized programs could be used for inference, but this could limit universality because there could be specific DRL inferences with extensions that require default DRL model policies, or at least are incompatible with programmatic policies. Thus, condition (R5) might be missed, but this needs further investigation. The same is true for symbolic policies.

Both fuzzy controllers and logical rules use training to represent a strategy, which introduces an approximation. Therefore, the requirement (R2.1) is not fulfilled.

The IBMDP method also appears to be promising, as it can generate understandable DTs that are verifiable, thus satisfying (R1) and (R4). This method can also represent entire policy models as DTs, thus satisfying (R2). It can be used during training to construct the DT, but the final version of the trained policy model also corresponds to an extracted DT, allowing it to be explained post-hoc as well, thus satisfying (R3).

Unfortunately, it does not explain the agent model directly with observation-action, but indirectly with state-action relations, so it does not satisfy (R1.1). Also, it uses adaptations, which allow all inputs and bounds to be explained exactly, but not explicitly uncovered areas, as other models do, as described previously. Therefore, (R2.1) is not satisfied. Although

the required modification of existing DRL methods is not model-specific, it may affect other potential modifications, so the requirement (R5) is not met.

Since there is no method that meets all the requirements that have been raised, a new method must be developed. For this purpose, the constraints are analyzed to fulfill all requirements.

5.2.2 Analysis of representations

Representing models by programs or DTs are both commonly understood self-explanations and allow formal verification, so they are appropriate choices because they satisfy (R1) and (R4). They can also both explain the entire policy model post-hoc and thus satisfy (R2) and (R3).

But neither of them explains the agent's model directly or accurately and thus do not satisfy (R1.1) or (R2.1) as described in Section 5.2.1. They also rely on modifications to the standard methods of DRL, so they do not satisfy (R5).

5.3 Contributions

In Section 5.2, XRL methods are described that could potentially fit but do not because they do not meet all the requirements of Section 5.1. Therefore, one idea is to use the equivalence description as described in Section 4.2. The main contribution to this problem of explainability of efficiently learned policies is that the presented approach of the NN2EQCDT algorithm, as described in [LV23a], directly represents the entire input-output behavior understandably. For this purpose, complete FF-DNNs are transformed into compressed DTs, as schematically shown in Figure 5.1, to improve explainability, interpretability, and verifiability. With the NN2EQCDT algorithm, all requirements are met as follows:

- The equivalence description of DNNs and DTs uses a left-sided linear transformation, but it is not compatible with the widely used DL framework PyTorch, which uses a right-sided linear transformation. Therefore, a conceptual algorithm for using a right-sided linear transformation and bias for ReLU analogous to that of Aytekin [Ayt22] is derived to directly use models learned with PyTorch.

- When using the equivalence transformation, the DT grows exponentially with branching. This problem is solved by lossless compression for smaller but equivalent models, which improves human interpretability.

- Dynamic compression significantly reduces the computation time and can also potentially reduce the inference time of the DT.

- There may be constraints inherent in the system that affect the model but are not considered in the transformation. Therefore, these are included as invariants in the satisfiability check to further compress the DT.

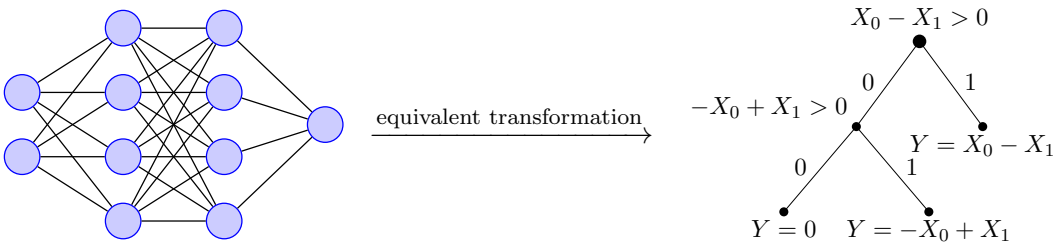


Figure 5.1: Schematic overview of the NN2EQCDT process. The DNN on the left represents the agent's actor model, which is learned by a standard DRL policy-based method that can be fully transformed directly into an equivalent and compressed DT. This transformation can be performed at any time step of the training or before simple inference. The DT thus provides exact traceability of input-output relationships, allowing an expert to indirectly extract the exact strategies learned.

- An implementation[1] is provided for transforming FF-DNN into equivalent compressed DTs and generating visualizations from DTs.
- Finally, the NN2EQCDT was shown to be able to accurately and directly explain DRL policy models by extracting one model from a benchmark problem and one from an ARL attack agent in a power grid scenario, i.e., in the context of CNIs.

In addition to these contributions, the NN2EQCDT algorithm has innovations over the EC-DT and the equivalence description algorithm, as described in Section 4.2.2. Rule validation is explicitly integrated into this one, and a new dynamic rule satisfiability check has been introduced to keep DTs small during generation and drastically reduce generation time.

The NN2EQCDT algorithm can convert entire policy models directly into DTs. Their explainability to humans is thereby increased by compression so that (R1), (R1.1), and (R2) are satisfied. Unlike the other methods, the NN2EQCDT algorithm does not extract a representation but transforms policy models with their weights directly into DTs, so the inference process is exactly the same as for the DNN models, so it also satisfies (R2.1). Domain experts can interpret parts of a DT and formulate invariant constraints for further compression and perform verifications so that (R4) is satisfied. The transformation of the NN2EQCDT algorithm can also be applied to the policy model at any time, i.e., during training and post-hoc, so that (R3) is satisfied. Finally, no changes are required to the standard policy-based DRL methods, since the interface is directly the DNN policy model, so (R5) is also satisfied.

[1] The implementation of the presented NN2EQCDT algorithm can be found at: `https://gitlab.com/arl-experiments/nn2eqcdt`

6 The NN2EQCDT algorithm

The presented NN2EQCDT algorithm efficiently transforms learnable FF-DNNs into compressed DTs to be able to explain them. For this purpose, in Section 6.1 it is shown how a simple FF-DNN with ReLU activation function in the form of a PyTorch model can be transformed into a DT using the effective weight matrices calculated with the NN2EQCDT algorithm. As a basis for this, in Section 6.2 the representation with the right-handed linear transformation is derived.

In the NN2EQCDT algorithm, the generation of a path starting from a node is dynamically terminated, as described in Section 6.3, if its assertion, along with all assertions along the path before it, is unsatisfiable. Also, further tree compression is described in Section 6.4. After the detailed description of the NN2EQCDT algorithm, its integration into the ARL framework is described in Section 6.5. Finally, the NN2EQCDT algorithm is further discussed in addition to the contributions in Section 5.3, especially with its limitations.

6.1 Decision tree construction with the NN2EQCDT algorithm

FF-DNNs with the ReLU activation function can be equivalently transformed to compressed DTs using the NN2EQCDT construction algorithm shown in Algorithm 3. It is strongly based on the equivalence description of Aytekin [Ayt22], as described in Section 4.2.2. However, it contains significant innovations, so it has already been published in [LV23a]. The algorithm generates DTs by iterative computing and joining subtrees with effective layer-wise filters from weight and bias matrices of FF-DNNs. It shows the access to the final effective filters and the calculation of the activation vector from the paths of the subtrees, as well as the conversion of the final rules into expressions and the compression of the whole tree. Here the algorithm is described in general and how it can be used. The individual components are explained in more detail in later sections.

The weight and bias matrices $\boldsymbol{W}_i$ and $\boldsymbol{B}_i$ from the FF-DNN are processed from the input to the output layer. These are used to compute rules that are used to add subtrees to the entire DT. In this way, a DT can be built dynamically while iterating the associated FF-DNN model. Starting from the second layer, the multiplication of the weight and bias matrices must take into account the position of the node to which the generated subtree is to be attached. This is done by applying the slope vector $\boldsymbol{a}$ to the current weight matrices. It represents the node position of the connection since it is the vector of decisions according to the ReLU activation function along the path from the root to the connection node.

When adding a node of a newly created subtree to the whole tree, each path from the root to the node in question is checked for satisfiability. If there can be no input, so that their evaluation of the DT takes that path, the node in question and thus further subtrees are discarded to keep the size of the DT minimal. Finally, the last checks are converted to expressions, and the DT can be further compressed by removing unnecessary checks since they are evaluated the same for all possible inputs.

Algorithm 3
NN2EQCDT algorithm for generating DTs by iterative computation with dynamic compression and connecting subtrees with effective layer-wise filters from weight and bias matrices of FF-DNNs. It shows the handling of accessing the final effective filters and computing the activation vector from the paths of the subtrees, as well as converting the final rules into expressions.

$\hat{\boldsymbol{W}} = \boldsymbol{W}_0$
$\hat{\boldsymbol{B}} = \boldsymbol{B}_0^\top$
$rules = \text{calc\_rule\_terms}(\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}})$
$T, new\_SAT\_leaves = \text{create\_initial\_subtree}(rules)$
$\text{set\_hat\_on\_SAT\_nodes}(T, new\_SAT\_leaves, \hat{\boldsymbol{W}}, \hat{\boldsymbol{B}})$
for $i = 1, \ldots, n-1$ **do**
    $SAT\_paths = \text{get\_SAT\_paths}(T)$
    for SAT_path in SAT_paths **do**
        $\boldsymbol{a} = \text{compute\_a\_along}(\text{SAT\_path})$
        $SAT\_leave = \text{SAT\_path}[-1]$
        $\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}} = \text{get\_last\_hat\_of\_leave}(T, SAT\_leave)$
        $\hat{\boldsymbol{W}} = (\boldsymbol{W}_i \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{W}}$
        $\hat{\boldsymbol{B}} = (\boldsymbol{W}_i \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{B}} + \boldsymbol{B}_i^\top$
        $rules = \text{calc\_rule\_terms}(\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}})$
        $new\_SAT\_leaves =$
        $\text{add\_subtree}(T, SAT\_leave, rules, invariants)$
        $\text{set\_hat\_on\_SAT\_nodes}(T, new\_SAT\_leaves,$
        $\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}})$
$\text{convert\_final\_rule\_to\_expr}(T)$
$\text{compress\_tree}(T)$

6.2 Derivation of the representation with right-handed linear transformation

DTs can be constructed from the effective weight matrices $\hat{\boldsymbol{W}}$ computed by spanning and joining subtrees through them. The algorithm for this is shown in Algorithm 4. It is first motivated and then explained by its construction.

Based on the NN2EQCDT algorithm, the linear transformation is performed with a left-hand multiplication of the weight matrix, as described in Section 4.2.2, but no implementation is given. From the requirement (R5) described in Section 5.1, it can be inferred that an XRL method must support standardized and widely used DL frameworks, such as PyTorch, in which DRL algorithms are also implemented so that they do not have to be modified for the transformation. Then existing models can also be efficiently reused in a quasi-standard format. However, PyTorch uses a right-handed rather than left-handed multiplication of the weight matrices [Fou23b] as follows:

$$\boldsymbol{Y}_\text{l} = \boldsymbol{W}_\text{l}^\top \boldsymbol{X} + \boldsymbol{B} \qquad \boldsymbol{Y}_\text{r} = \boldsymbol{X}\boldsymbol{W}_\text{r}^\top + \boldsymbol{B}$$

To construct a DT from a Pytorch model consisting of linear layers with bias and applying the activation function $\sigma = \text{ReLU}$ between them, the layer-wise effective weight matrices $\hat{\boldsymbol{W}}_i$ must be computed using the right-handed linear transformation with bias, as described in Eq. (6.1) based on [Ayt22]. Here, the activation function is performed by element-wise multiplication of the activation slopes by the weight matrices. The activation vector $\boldsymbol{a}$ must be transposed and repeated k times so that the multiplication is equal to the size of the matrices to which it is applied. Thus, fully qualified, the expression $[(\boldsymbol{a}^\top)_{\times k}]$ is used, as in the algorithms. In the following equations, however, it is written simply as $\boldsymbol{a}$ when repeated analogously to [Ayt22], that is, $\boldsymbol{a} \coloneqq [(\boldsymbol{a}^\top)_{\times k}]$.

$$
\begin{aligned}
\hat{\boldsymbol{W}}_i^\top &= \sigma(\boldsymbol{x}_{i-1}\boldsymbol{W}_{i-1}^\top + \boldsymbol{B}_{i-1})\boldsymbol{W}_i^\top + \boldsymbol{B}_i \\
&= \sigma((\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top)^\top)\boldsymbol{W}_i^\top + \boldsymbol{B}_i \\
&= (\boldsymbol{a}_{i-1} \odot (\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top)^\top)\boldsymbol{W}_i^\top + \boldsymbol{B}_i \\
&= ((\boldsymbol{a}_{i-1}^\top \odot (\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top))^\top)\boldsymbol{W}_i^\top + \boldsymbol{B}_i \\
&= (\boldsymbol{W}_i(\boldsymbol{a}_{i-1}^\top \odot (\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top)))^\top + \boldsymbol{B}_i \\
&= ((\boldsymbol{W}_i^\top \odot \boldsymbol{a}_{i-1}^\top)^\top(\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top))^\top + \boldsymbol{B}_i \\
&= ((\boldsymbol{W}_i \odot \boldsymbol{a}_{i-1})(\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top))^\top + \boldsymbol{B}_i \\
&= (((\boldsymbol{W}_i \odot \boldsymbol{a}_{i-1})(\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top)) + \boldsymbol{B}_i^\top)^\top
\end{aligned}
\tag{6.1}
$$

The recursive form in Eq. (6.1) can be used to formulate a general closed equation as shown in Eq. (6.2) based on [Ayt22]. It is equivalent to the right-handed linear transformation with bias and ReLU activation function.

$$
\begin{aligned}
\text{NN}(\boldsymbol{x}_0) &= (\dots((\boldsymbol{W}_1 \odot \boldsymbol{a}_0)(\boldsymbol{W}_0\boldsymbol{x}_0^\top + \boldsymbol{B}_0^\top) + \boldsymbol{B}_1^\top)\dots)^\top \\
&= (\dots(\underbrace{(\boldsymbol{W}_1 \odot \boldsymbol{a}_0)\boldsymbol{W}_0}_{\hat{\boldsymbol{W}}_{1,a_0}}\boldsymbol{x}_0^\top + \underbrace{(\boldsymbol{W}_1 \odot \boldsymbol{a}_0)\boldsymbol{B}_0^\top + \boldsymbol{B}_1^\top}_{\hat{\boldsymbol{B}}_{1,a_0}})\dots)^\top
\end{aligned}
\tag{6.2}
$$

The corresponding algorithm for a simple FF-DNN with linear transformation, bias, and ReLU activation function is shown in Algorithm 4. The subscript j of $\hat{\boldsymbol{W}}_j \in \mathbb{R}^{1\times fs}$ with $\boldsymbol{x}_0 \in \mathbb{R}^{bs\times fs}$ and a batch size of $bs \geq 1$ and a feature size of $fs \geq 1$ refers to the j-th row of the current $\hat{\boldsymbol{W}} \in \mathbb{R}^{k\times fs}$, but the index $i+1$ of $\boldsymbol{W}_{i+1}$ refers to the weight matrix of the $i+1$-th layer and not to a row. The same is true for the bias matrix. $[(\boldsymbol{a}^\top)_{\times k}]$ means that the transposed vector $\boldsymbol{a}^\top \in \mathbb{R}^{m_i\times 1}$ is repeated line by line k times.

Algorithm 4 is an innovation over Algorithm 1 in that it supports bias matrices and explicitly specifies the dimensions of the matrices and the repetition of the $\boldsymbol{a}$ vector to make them easier to understand. More importantly, however, the algorithm uses right-handed multiplication as derived from Eq. (6.2) instead of left-handed multiplication as derived from Eq. (4.5) to support the direct use of FF-DNN models in PyTorch format, thus satisfying requirement (R5) described above.

To better understand the application of Algorithm 4, a simple example of converting the XOR function into a DT is given in Figure 6.1. The XOR function is represented by the

Algorithm 4
Algorithm for computing effective weight matrices with right-handed linear transformation and bias using ReLU activation function, based on [Ayt22]

$\hat{\boldsymbol{W}} = \boldsymbol{W}_0$
$\hat{\boldsymbol{B}} = \boldsymbol{B}_0^\top$
for $i = 0, \ldots, n-2$ **do**
  $\boldsymbol{a} = [\,]$
  for $j = 0, \ldots, m_i - 1$ **do**
    if $(\hat{\boldsymbol{W}}_j \boldsymbol{x}_0^\top + \boldsymbol{B}_j^\top)^\top > 0$ **then**
      $\boldsymbol{a}$.append(1)
    else
      $\boldsymbol{a}$.append(0)
  $\boldsymbol{W}_{i+1} \in \mathbb{R}^{m_i \times k}, \boldsymbol{a} \in \mathbb{Z}_2^{m_i}$
  $\hat{\boldsymbol{W}} = (\boldsymbol{W}_{i+1} \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{W}}$
  $\hat{\boldsymbol{B}} = (\boldsymbol{W}_{i+1} \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{B}} + \boldsymbol{B}_{i+1}^\top$
return $(\hat{\boldsymbol{W}} \boldsymbol{x}_0^\top + \hat{\boldsymbol{B}})^\top$

following weight matrices of the linear layers without bias, as in the example for the EC-DT algorithm described in Section 4.2.1. Inner identity weight matrices were added to later show the linkage of the splitting points of the inner layers in the transformed DT. Therefore, the forward operations are computed first to show that the input indeed maps the XOR function.

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad \boldsymbol{W}_0 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad \boldsymbol{W}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \boldsymbol{W}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \boldsymbol{W}_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 \boldsymbol{W}_0^T = \begin{bmatrix} 0 & 0 \\ -1 & 1 \\ 1 & -1 \\ 0 & 0 \end{bmatrix} \qquad \boldsymbol{x}_{1,a} = \mathrm{ReLU}(x_1) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} = \boldsymbol{x}_{2,a} = \boldsymbol{x}_{3,a} = \boldsymbol{x}_{i \in \{1,2,3\},a}$$

$$\boldsymbol{x}_4 = \boldsymbol{x}_{i \in \{1,2,3\},a} \boldsymbol{W}_0^\top = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \boldsymbol{x}_{4,a} = \boldsymbol{y}$$

The weights without the identity matrices are used to calculate the following effective weight matrices. These are then used to calculate the rules for the split points and to span the tree according to the activation paths.

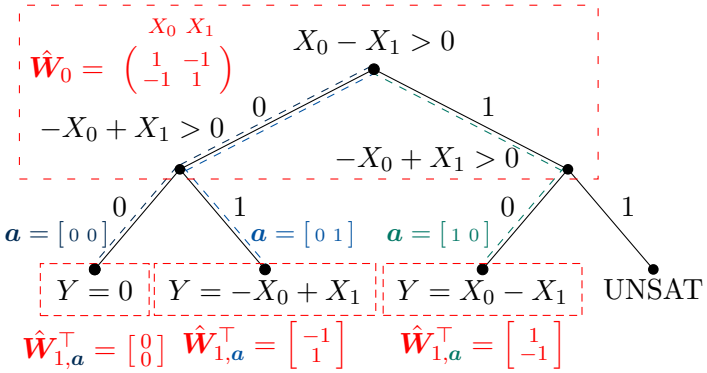


Figure 6.1: Simple example of a DT representing an XOR function constructed with Algorithm 4 without bias based on the XOR example for the EC-DT algorithm as described in Section 4.2.1. Each coefficient of a row of $\hat{\boldsymbol{W}}_i$ is linearly expanded and used as a split point rule with the activation function ReLU*. After a layer, only the $\hat{\boldsymbol{W}}_{i+1,\boldsymbol{a}}$ with the respective previous activations $\boldsymbol{a}$ are used for branching.*

$$\hat{\boldsymbol{W}}_0 = \boldsymbol{W}_0 \qquad \boldsymbol{a}_{0,\boldsymbol{x}_{0,0}} = \mathrm{ReLU}\left(\begin{bmatrix} \underbrace{\begin{bmatrix} 1 & -1 \end{bmatrix}}_{\hat{\boldsymbol{W}}_{0,0}=\boldsymbol{W}_{0,0}} \cdot \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\boldsymbol{x}_{0,0}} & \underbrace{\begin{bmatrix} -1 & 1 \end{bmatrix}}_{\hat{\boldsymbol{W}}_{0,0}=\boldsymbol{W}_{0,1}} \cdot \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\boldsymbol{x}_{0,0}} \end{bmatrix}\right) = \begin{bmatrix} 0 & 0 \end{bmatrix} \in \mathbb{R}^{2\times 1}$$

$$\hat{\boldsymbol{W}}_{3,\boldsymbol{x}_{0,0}} = (\boldsymbol{W}_3 \odot [(\boldsymbol{a}_{0,\boldsymbol{x}_{0,0}})_{\times 1}])\hat{\boldsymbol{W}}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix} \qquad \boldsymbol{y}_{\boldsymbol{x}_{0,0}} = (\hat{\boldsymbol{W}}_{3,\boldsymbol{x}_{0,0}} \boldsymbol{x}_{0,0}^\top)^\top = \begin{bmatrix} 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} = [0]$$

$$\boldsymbol{a}_{0,\boldsymbol{x}_{0,1}} = \mathrm{ReLU}\left(\begin{bmatrix} \underbrace{\begin{bmatrix} 1 & -1 \end{bmatrix}}_{\hat{\boldsymbol{W}}_{0,0}=\boldsymbol{W}_{0,0}} \cdot \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{\boldsymbol{x}_{0,1}} & \underbrace{\begin{bmatrix} -1 & 1 \end{bmatrix}}_{\hat{\boldsymbol{W}}_{0,0}=\boldsymbol{W}_{0,1}} \cdot \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{\boldsymbol{x}_{0,1}} \end{bmatrix}\right) = \begin{bmatrix} 0 & 1 \end{bmatrix} \in \mathbb{R}^{2\times 1}$$

$$\hat{\boldsymbol{W}}_{3,\boldsymbol{x}_{0,1}} = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \end{bmatrix} \qquad \boldsymbol{y}_{\boldsymbol{x}_{0,1}} = (\hat{\boldsymbol{W}}_3 \boldsymbol{x}_{0,1}^\top)^\top = \begin{bmatrix} -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = [1]$$

$$\boldsymbol{a}_{0,x_{0,2}} = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad \boldsymbol{a}_{0,x_{0,3}} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$\hat{\boldsymbol{W}}_{3,\boldsymbol{x}_{0,2}} = \begin{bmatrix} 1 & -1 \end{bmatrix} \qquad \hat{\boldsymbol{W}}_{3,\boldsymbol{x}_{0,3}} = \begin{bmatrix} 0 & 0 \end{bmatrix} \qquad \boldsymbol{y}_{\boldsymbol{x}_{0,2}} = [1] \qquad \boldsymbol{y}_{\boldsymbol{x}_{0,3}} = [0]$$

Each coefficient of a row of $\hat{\boldsymbol{W}}_i$ is linearly expanded and used as a split point rule with the activation function ReLU. After a layer, only the $\hat{\boldsymbol{W}}_{i+1,\boldsymbol{a}}$ with the respective previous activations $\boldsymbol{a}$ are used for branching.

In Algorithm 4, the next weight matrix is needed to calculate $\hat{\boldsymbol{W}}$. The same applies here as well as in the following for the bias matrices. The iteration index can be equivalently transformed from $i+1$ to i by using iteration with **for** $i = 1, \ldots, n-1$ **do**. Thus, the last effective weight matrix is needed to compute the current weight matrix and the next subtree to be attached. This is used in the NN2EQCDT algorithm in Algorithm 3 to be able to use the last effective weight matrices instead of the next. To access them in the current iteration, they are stored in the iteration before by appending them to all SAT nodes of the subtrees they span. In addition to the last effective and current weight matrix, the activation vector $\boldsymbol{a}$ is also needed to calculate the new effective weight matrix.

In the concept Algorithm 4, it is computed by using the input to branch to different activations. When converting the concept to an implementation with a dynamic design, as conceptualized in Algorithm 3, the activation vector is required for each temporary SAT node for which the next effective weight and bias matrices are computed and attached as a subtree. Since the activation vector corresponds to the branches of a path, it is computed along this path in the DT, as seen in Figure 6.1.

6.3 Dynamic path checking when adding subtrees

The algorithm in Algorithm 4 nevertheless generates potentially large DTs because it grows exponentially, as described in Section 4.2.2. There may also be paths in the DT that cannot be taken because an equivalence description need not make any statement about them. It only cares about the exact same input-output relationship for each input point.

Such paths cannot be taken because split-point rules evaluate only one output for all possible inputs. The input range of split-point rules of such nodes is reduced to a subrange with only one branch because in the hierarchy of the DT split-point rules of nodes above such a node already trap input points that would otherwise lead to a branch of this split-point rule. This is consistent with the behavior of the inference of the DNN model. If an input region always allows a parameter of a layer to be evaluated as negative, then it is mapped to zero by the activation function ReLU, which, when multiplied by a parameter of the next linear layer, gives the same output, zero, so that this layer can also branch in only one direction, as in the transformed DT.

In a DT spanned by the algorithm in Algorithm 4, there may be regions that are invalid due to conflicting categorizations. For example, the split point rule $x > 0$ and its inverse $x \leq 0$ contradict each other, so they are not jointly satisfiable. When such jointly unsatisfiable split point rules occur along a path in a tree, the path is invalid from that point on.

When a subtree is added to the whole DT, the joint satisfiability of all path rules is checked to avoid unnecessary calculation of additional paths that cannot be satisfied. If a path is not satisfiable from a particular node, there is no input where the evaluation of the DT follows the path from that node. The path is then terminated with a UNSAT node, as exemplified in Figure 6.1. Since the path cannot be traced further, further checks and associated nodes and subtrees are not required. Consequently, node concatenation and subtree computation can be terminated from this node. In this way, the DT is dynamically compressed in the design phase, but is still equivalent to the input FF-DNN model, since only unreachable checks are omitted.

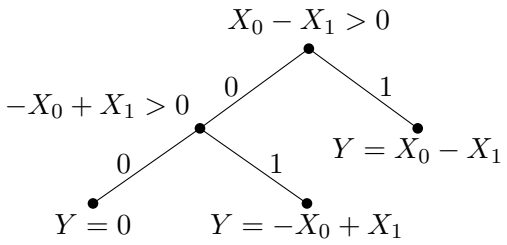


Figure 6.2: Simple example of a compressed DT using the DT example in Figure 6.1 by removing the right check $-X_0 + X_1 > 0$, which evaluates to false for all inputs, since the root check $X_0 - X_1 > 0$ evaluates to true in this branch.

In addition to path rules, other constraints, such as input ranges or output checks for input ranges, can be used as invariants by expressing them as assertions. This allows the DT to be further compressed while maintaining equivalence since further potentially unnecessary nodes can be omitted due to the invariants. All related assertions, such as path assertions and general input domain assertions, can be written in SMT format and are checked for satisfiability together with the SMT solver Z3 as described in Section 3.7.

Since path generation can be dynamically stopped at certain nodes, entire subtrees may not be calculated. This can compress DTs and increase the overall computation time while maintaining an equivalent representation.

6.4 Further tree compression

In addition to pruning a DT as it is created, it can be further compressed by removing checks in it that are evaluated the same for their entire direct input space and are therefore not needed to distinguish inputs from one another.

When a DT is created while its paths are dynamically checked for satisfiability, it can have UNSAT nodes as leaves, as seen in Figure 6.1. This example can be compressed, as seen in Figure 6.2, by removing the right-hand check $-X_0 + X_1 > 0$, which evaluates to false for all inputs, since the root check $X_0 - X_1 > 0$ evaluates to true in this branch.

In any case, the rule of a parent node to a UNSAT node is evaluated on the further path of the non-UNSAT node, since the rules and their evaluations of the nodes preceding it in the path reduce the input space to this evaluation region. Since there is otherwise no input to evaluate, the rule checking of a parent node to a UNSAT node can be omitted. Therefore, the entire parent node of a UNSAT node can be replaced by the non-UNSAT child node and its associated subtree, as seen in an example in Figure 6.2, without the DT losing accuracy compared to the FF-DNN. This operation is therefore consistent with the goal of equivalent transformation of a FF-DNN into a DT. The process of compressing a UNSAT node is shown in Figure 6.3.

Suppose a check $\sum b_i x_i > 0$ in a node always evaluates to false, e.g., to the left branch, because evaluating to true, e.g., to the right branch, is unsatisfiable because all possible inputs are already caught away, as described in Section 6.3. Therefore, the rule and hence the node can be omitted. In this case, the left subtree with the check $\sum c_i x_i > 0$ in the root node is

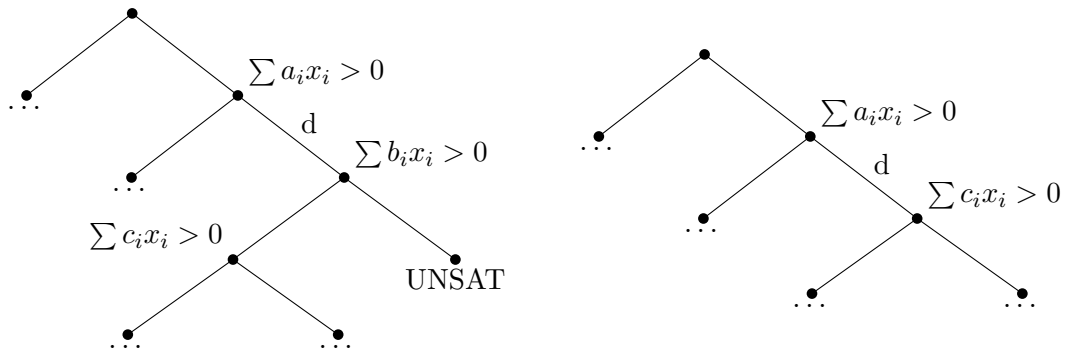


Figure 6.3: Compression method within an SAT-checked DT by replacing a parent of a UNSAT node with its non-UNSAT child node and a connected subtree.

connected to the node of the original check $\sum b_i x_i > 0$. In this case, this is the node connected to the right path d from the node with the check $\sum a_i x_i > 0$.

6.5 ARL framework integration

This work will be part of the ARL framework. Its architecture is proposed in [Vei23], as shown in Figure 6.4. It is not explained in detail here, but the embedding of this work and why it is an integral part of the proposed architecture is described.

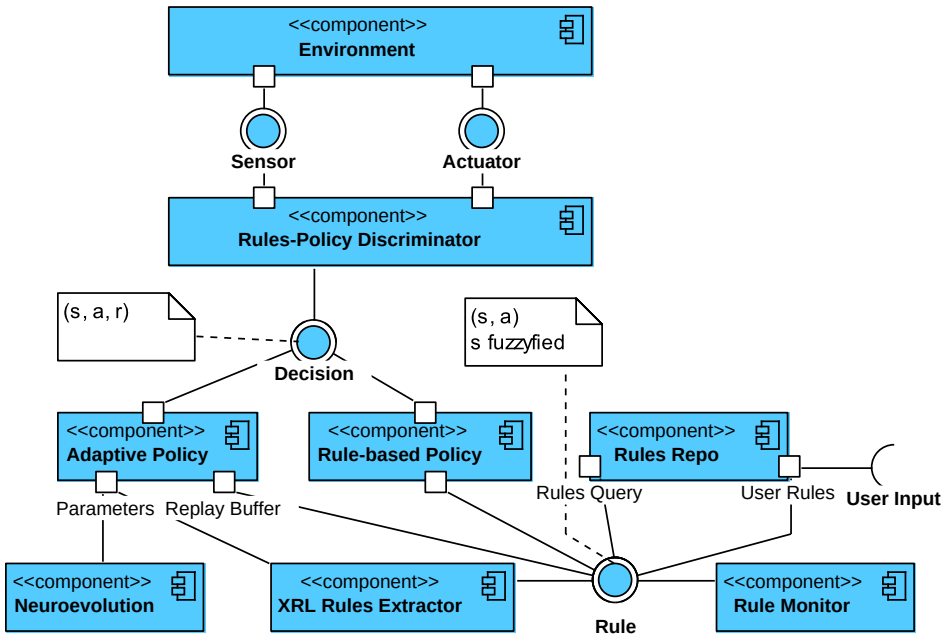


Figure 6.4: ARL architecture [Vei23]

The concept of equivalent transformation of FF-DNNs to DTs can be used for or at least in the *XRL Rules Extractor*. The idea of this component is to extract rules from the actor

model of the DRL policy. The policy is adaptive in that it is continuously trained and adjusts its weights and biases accordingly. For such a scenario, the extraction could be performed regularly. The extracted rules are stored in the *Rules Repo* component, as shown in Figure 6.5, and used together with rules from user input in the *Rules-based Policy* component. In each action step, both policies propose a *Decision*, but only one is selected. When the adaptive policy proposal is chosen, its models are batch trained and therefore adapted.

The concept of equivalent transformation from FF-DNNs to DTs can be used for or at least in *XRL Rules Extractor*. The idea of this component is to extract rules from the actor model of the DRL policy. The policy is adaptive in the sense that it is continuously trained and adjusts its weights and biases accordingly. In such a scenario, the extraction could be performed at regular intervals. The extracted rules are stored in the *Rules Repo* component, as shown in Figure 6.5, and used together with rules from user input in the *Rules-based Policy* component. In each action step, both policies suggest a decision, but only one is selected. When the adaptive policy is selected, its models are batch trained and thus adapted.

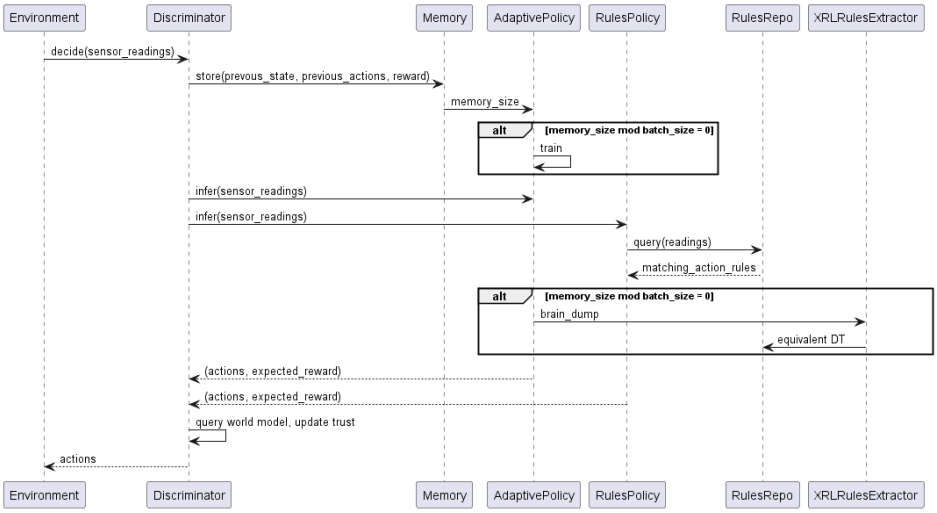


Figure 6.5: Activity diagram for training and self-explaining of an ARL agent, from [VL23]

The idea is that this loop of action decision, adaptive training, rule extraction, and static rule selection should on the one hand stabilize the training, but on the other hand, also shield the actual decision from safety endangering actions. Rule extraction is an integral part of this loop. Without the extraction, the shielding could also work, but the static rules would not be able to be updated, which could reduce the long-term performance of the rule set optimization, since then only the Adaptive Policy is updated and potentially cannot be used in each step.

Shielding is especially important when the agent explores and adaptively updates its *DRL* policy, otherwise, it might choose actions from harmful regions. The *Rules-Policy Discriminator* component must then select the potentially harmless decision from the rules-based policy. It decides based on the computed confidence and confidence level of the proposed decisions. The confidence level is a value that describes how confident the particular policy is about the proposed reward using an internal world model. The confidence level describes the extent to which the estimated reward and confidence are consistent with the safety constraints. This value contains more external information and is history-based.

In addition to the shielding loop, the extracted rules or directly the transformed DT of the NN2EQCDT algorithm can be used to explain and validate the learned behavior of the Adaptive Policy during continuous learning and can be transferred to the rule set only if a human operator accepts them [VL23].

6.6 Discussion

Although the NN2EQCDT algorithm satisfies all the requirements described in Section 5.3, this does not mean that other XRL methods are no longer needed. As described in Section 3.6, they can focus on other things, such as other levels of explainability to other audiences. They can also explain the agent's behavior by answering questions such as why strategies were chosen or why not, which might help people understand the agent's thought process. Some methods focus on other DRL components such as rewards, tasks, or states, as described in Section 4.1.

In particular, the policy-based self-explainable methods and within them, the PRL and to some extent the other DT methods, allow the extraction of generalizations in the form of DSLs or DTs that are also verifiable. These can be better explained by people, especially experts because they focus more on general, causal explanations. However, they may have the disadvantage of not being exact explanations.

The NN2EQCDT algorithm also has some limitations in its current form. It can only transform FF-DNNs with linear layers and ReLU-action functions into DTs. However, this may be sufficient since, as described in Section 3.3.2, they can approximate any measurable Borel function. They are also commonly used as DRL policy models.

Despite the compression, the resulting DT can still become very large, which reduces its explainability to humans. While the NN2EQCDT algorithm explains the exact observation-action relation and thus allows for exact traceability, it does not provide potentially more human-understandable explanations about the actual strategies and why, or why not, and how they were learned.

7 Evaluation Scenario

In this chapter a scenario is described, which is used for the later evaluation of the NN2EQCDT algorithm to explain the learned strategy of a DRL policy. To demonstrate, how a behavior of an agent, learned for an objective in a power grid setting, that is described in Section 7.1, can be explained with the presented method, an experiment with a simple voltage attack agent in a small power grid is conducted in Section 7.2. To incorporate the ARL idea of symbiosis effects, autocurriculum learning was applied by additionally using a simple voltage defender agent.

This chapter describes a scenario used for later evaluation of the NN2EQCDT algorithm to explain the learned strategy of a DRL policy. To demonstrate how behavior of an agent can be explained using the presented method, an experiment is conducted with a simple agent in a small power grid in Section 7.2. Here, the agent has the one objective to destabilize the voltage in a power grid environment described in Section 7.1. To incorporate the ARL idea of symbiotic effects, autocurriculum learning was applied by additionally using a simple voltage defender agent.

7.1 Power grid setting

The power grid used for the scenario under study is a simple line connected to an external grid via a transformer. It is specified as a panda power grid, which is shown in Figure 7.1 and can be found together with the MIDAS scenario [Wol+23], the runfile, and other configuration files at [LV23b].

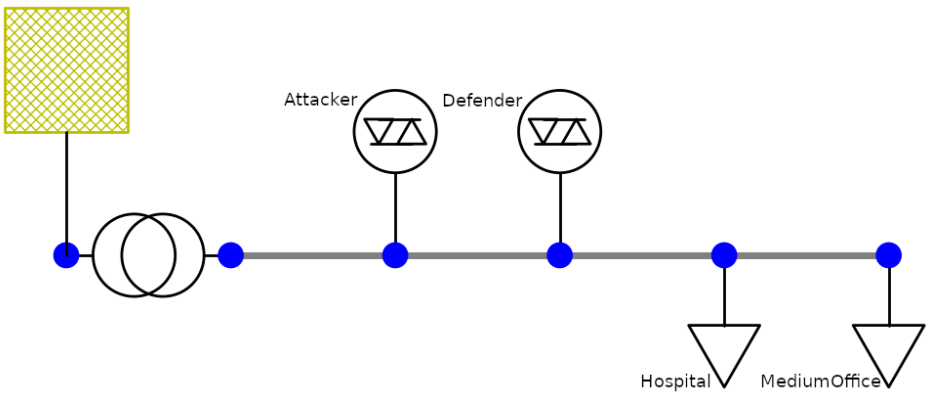


Figure 7.1: Simple network used with buses (blue dots) connected by only one (gray) line. The yellow network structure at the top left represents the external network. The other symbols from left to right: a transformer and two synthetic generators as well as two loads with different load profiles.

The blue dots represent the buses connected by (gray) lines. The transformer is located between the first two buses. It is a *grid coupling transformer* [Sch12, p.371] because it connects with 63 MVA the voltage levels 100 kV and 20 kV from the external grid and the simple, connected grid.

The next two buses further down the line each have a synthetic reactive power injector connected as a generator. Finally, a hospital and a MediumOffice are connected to the last two buses of the line as two loads. In Section 3.1.2, it is described how these components work and are used in a power grid.

The loads are not static but dynamic, have different load profiles, and therefore consume different amounts of electricity at different times. For the load profiles, real commercial data from the project Open Energy Data Initiative (OEDI) [US 23] as of 01.05.2017 is used. The aggregated load is visualized in Figure 7.2 for the profiles used.

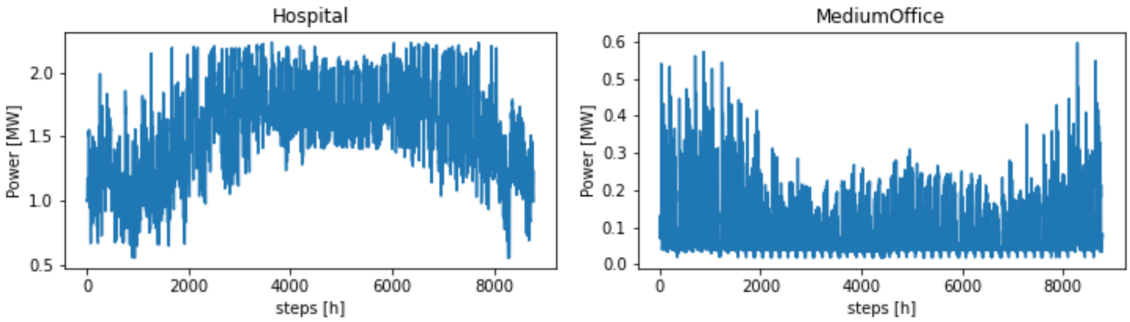


Figure 7.2: Visualization of aggregated load profiles for hospitals and medium office buildings [Bal23], this commercial data comes from the OEDI [US 23] project

The power system is loaded under the MIDAS scenario and the setpoints of the synthetic reactive power injections are controlled by palaestrAI. The actual bus voltages are calculated by pandapower using solvers for PFC as described in Section 3.1.3.

7.2 Simple voltage attacking scenario

A scenario for a voltage attack is set up in this power grid. For this purpose, two agents are deployed. The main agent, the attacker, has the goal to destabilize the voltage level as much as possible, i.e., to deviate the voltage level as much as possible from 1.0 p.u.. The other agent, the defender, aims to keep the voltage level as close as possible to 1.0 p.u.. This learning situation with two agents with such conflicting goals is an autocurriculum, as described in Section 3.5.5. This scenario will be published in [VL23]. The following is taken from or relates to that paper.

For the ARL use case, it was adapted to be more realistic. The agents do not receive the actions directly from the other agents. The idea to fulfill the autocurriculum requirement is that an agent learns the already learned and integrated strategies of the other agents indirectly through observations at other measurement points.

A bell-shaped curve is used for both objectives, with a center of 1.0 p.u.: The defender's maximum reward was 1.0 p.u., while the attacker used the reverse curve, with maximum reward at $V < 0.8$ p.u.) resp. $V > 1.1$ p.u., as shown in Figure 7.3.

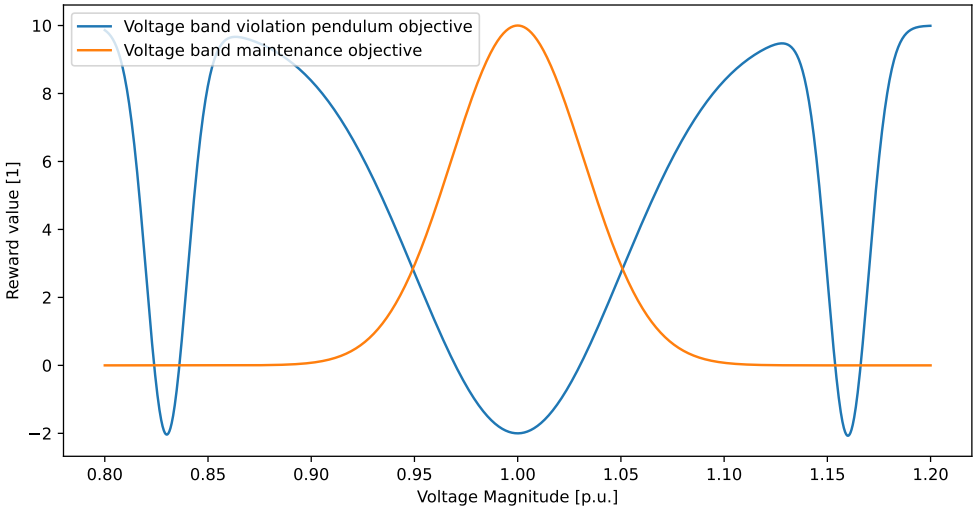


Figure 7.3: Visualization of the objectives for maintaining the voltage bandand exceeding it in the form of a pendulum with rewards plotted against voltage magnitudes

Consider the reward function

$$g(x = \frac{\sum_{i=1}^{|\boldsymbol{V}|} V_i}{|\boldsymbol{V}|}, A, \mu, C, \sigma) = A \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} - C\right) \ , \tag{7.1}$$

where $\boldsymbol{V}$ are voltages at the observed "victim buses" to which the dynamic loads are connected. The parameters A, μ, C, and σ shape the curve, so that is defined as:

$$\begin{aligned} reward_{attacker}(x = \frac{\sum_{i=1}^{|\boldsymbol{V}|} V_i}{|\boldsymbol{V}|}) =& g(x, A = -12.0, \mu = 1.0, C = -10.0, \sigma = -0.05) \\ &+ g(x, A = -12.0, \mu = 0.83, C = 0.0, \sigma = 0.01) \\ &+ g(x, A = -12.0, \mu = 1.16, C = 0.0, \sigma = 0.01) \end{aligned} \tag{7.2}$$

$$reward_{defender}(x = \frac{\sum_{i=1}^{|\boldsymbol{V}|} V_i}{|\boldsymbol{V}|}) = g(x, A = 10.0, \mu = 1.0, C = 0.0, \sigma = 0.032) \ . \tag{7.3}$$

Agents are trained using the DDPG reinforcement learning algorithm, as described in Section 3.5.3. The optimization algorithm Adam, as described in Section 3.3.4.2, was used to optimize the model with a learning rate of $\alpha = 1\text{e}{-4}$, as described in Section 3.3.4.

7.3 Visual inspection of agents behavior

In Figure 7.4, the setpoints in MVar for the attacker agent are plotted for the time range given above. It can be seen that the attacker agent trained with the DDPG algorithm first

uses high setpoints, which are capped at 1.5 MVar in the training phase, then explores and finally tries to use low setpoints capped at −1.5 MVar on May 13. This low setpoint is always used in the test phase as well.

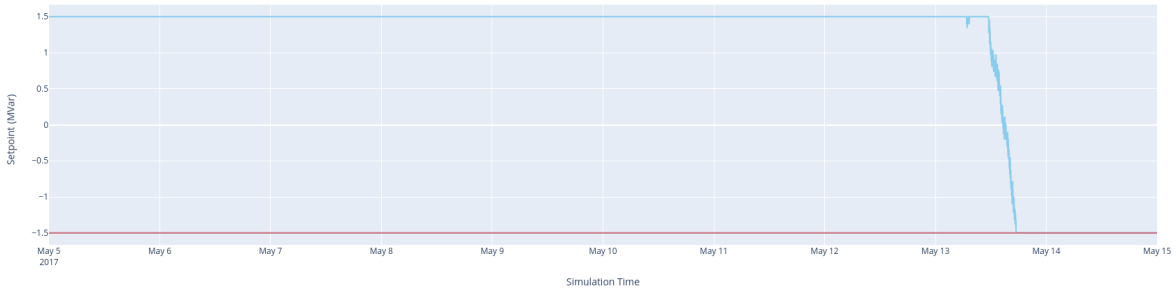


Figure 7.4: Attacker setpoints in the training and test phases of the simple voltage attack scenario, learned for a reactive power setpoint box of [−1.5 MVar, 1.5 MVar]. The blue line is for the training phase, and the red line is for the testing phase (they overlap at the bottom).

In Figure 7.5, the setpoints are plotted in MVar for the attacker agent for the time range specified above. The defender agent uses high setpoints that are pruned to 2 MVar during the training and testing phase.

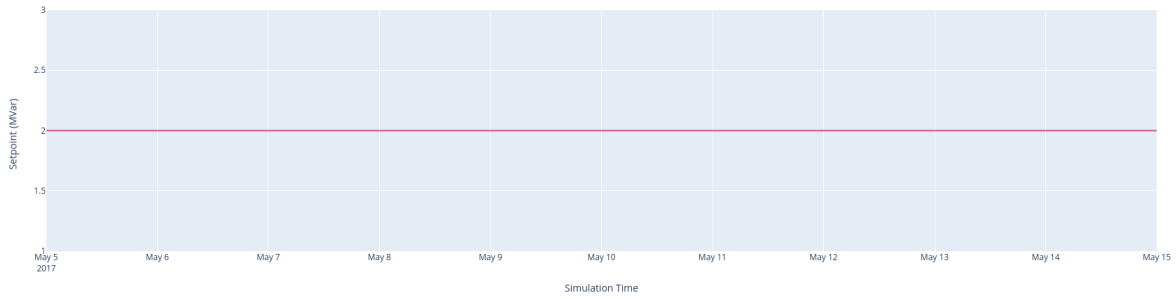


Figure 7.5: Defender setpoints in the train and test phase of the simple voltage attack scenario, learned for a reactive power setpoint box of [−1.5 MVar, 1.5 MVar]. The blue line is for the train and the red line is for the test phase (they overlap at the bottom).

The voltage magnitudes of buses 4 and 5 are shown in Figure 7.6 for the training and test phases. One can visually see that the bus voltages are always in [0.92 p.u., 1.02 p.u.] and mostly below 1 p.u.. It makes sense, then, that the defender agent has learned to set its actuator setpoint as high as possible to inject as much reactive power into the network as possible. The attacker agent, on the other hand, tries to consume as much reactive power as possible to further lower the voltage level.

The noise and oscillatory behavior of the voltage level may come from the profiles of the loads connected to buses 4 and 5, which are described in Section 7.1. These load profiles were chosen because they are relatively contrarian. The intent for such a selection was that it would help with learning oscillatory behavior from the attacker agent, but no load profile combination could accomplish this.

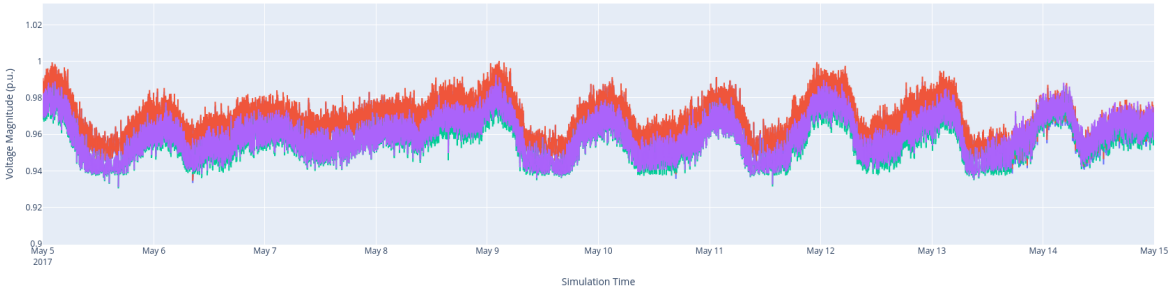


Figure 7.6: Voltage magnitudes of buses 4 and 5 in p.u. *of the example power system used are colored in blue and red in the training phase and in green and purple in the test phase*

The reward levels achieved by the attacker agent according to its voltage band pendulum objective, as shown in Figure 7.3, are shown in Figure 7.7 for the scenario. Comparing this to the voltage values of Figure 7.6, it can be seen that the reward changes are inverted voltage value changes. In this case, the agents have little influence.

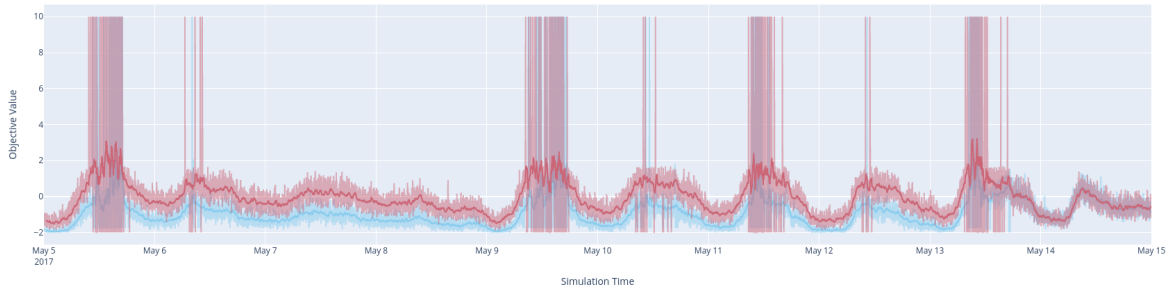


Figure 7.7: Objective values for the voltage attacker agent calculated by applying the stored rewards to the voltage defender objective function. The light red line represents the objective values and the dark red line represents the objective average over 100 values

7.4 Scenario adaption

The first idea was to use a standard benchmark power grid, like the Cigre MV grid [Tas14; Fra23] to create a scenario instead of a simple line. Essentially, this involves two feeders connected via an external network, as shown in Figure 7.8.

This effectively results in a single line of sensors and actuators, so the power network can be simplified into one. This was done, as described in Section 7.2, to avoid unnecessary complexity and bugs and because a custom pandapower network had to be used due to implementation details.

7.5 Conclusion and Discussion

A scenario is described here that will be used for later evaluation of the NN2EQCDT algorithm. It includes a description of the power system and its components and dynamics. The loads

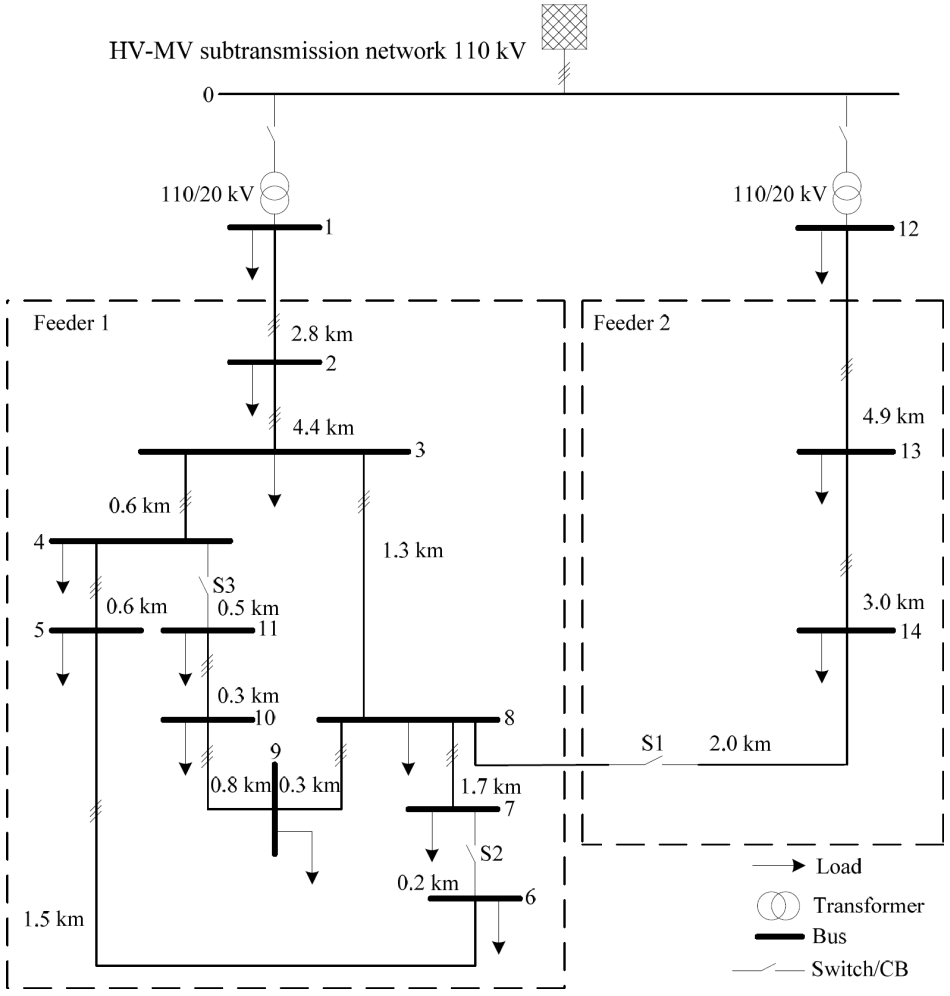


Figure 7.8: Cigre MV power grid network for benchmarking purposes [Tas14; Fra23]

are dynamic, i.e., they depend on the simulation time. The attacker and defender agents control the synthetic reactive power injection to manipulate the voltage level. They learn based on voltage band violation pendulums and maintenance objectives, which are in an autocurriculum. They have learned a fairly simple but successful strategy for each according to their objectives.

The learned oscillating strategy, as shown in Figure 7.9 [VWU23; Vei+22], should be reproduced to show that the presented method is also able to explain a more complex strategy.

However, such a more complex oscillatory strategy could not be replicated by learning an agent. Why this is so needs further investigation. It is also important to consider the limitations included, such as the small size of the DNN model, and the limitations described in Section 6.6, such as the simple DNN architecture with only linear and ReLU layers.

One idea why this might not have worked is that the attacker and the defender always act at the same time step, which means that they both always observe the applied actions of the other agent as an influence on their observations. Thus, they cannot each observe

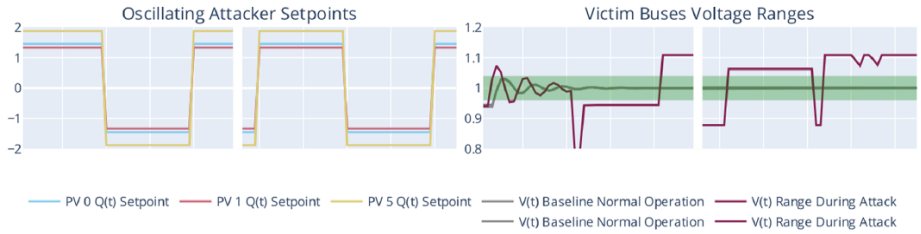


Figure 7.9: Setpoints of the oscillating attacker as well as the obtained voltage magnitudes, from [VWU23]. It should be noted that the diagram contains an intentional time gap between 00:50 and 03:15.

only the influence of their actions. The autocurriculum property could still be preserved if a *taking-turn* strategy is used for the agents, i.e., the agents act sequentially rather than in parallel, so that the influence of the action might be better learnable.

Another idea is that the agents are not faced with new challenges because they have already learned simple but quite effective strategies. To let them explore more, a kind of self-play strategy as described in Section 3.5.6 could be evaluated to be implemented in the agents of the learning phase.

8 Evaluation

8.1 NN2EQCDT evaluation

In this section, the NN2EQCDT algorithm is evaluated using a simple benchmark scenario. This has already been published in [LV23a].

8.1.1 Application of to simple model

A simple controller model was trained using the DDPG algorithm as described in Section 3.5.3, trained for the objective of reaching the hill with a car in the MountainCarContinuous-v0 environment (MCC) [Moo90; Fou23a].

Originally, an actor model as described in Section 3.5.3 and shown in Listing 8.1 was trained with a larger hidden size of $hid = 64$. Since it is not necessary and more difficult to further analyze a model of this size, a student model with $hid = 8$ and MSE loss only in the relevant region of $x \in [-1.2, 0.6]$ and $y \in [-0.7, 0.07]$ and a step size of 0.1 was distilled from the larger model. It was visually found to perform about the same as the larger model.

```
nn.Sequential(
    nn.Linear(2,   hid, bias=True), nn.ReLU(),
    nn.Linear(hid, hid, bias=True), nn.ReLU(),
    nn.Linear(hid,   1, bias=True)
)
```

Listing 8.1: Actor model in PyTorch trained with the DDPG algorithm with variable hidden size for the MCC

The smaller student model was then transformed into an equivalent compressed DT using the NN2EQCDT algorithm from Algorithm 3, as shown in Figure 8.1. DTs are represented by networkx graphs that can be plotted with pyvis, as shown in Figure 8.2.

The different regions of a DT can be easily separated for 2D inputs. If the expressions for each input are to be visualized, they can be evaluated for the corresponding decision region and displayed as a third dimension, as shown in Figure 8.2. The points for the 2D regions (x and y) are obtained by implicitly plotting with sympy. The values for the z dimension are evaluated for each x and y point by the final expression and then drawn as a scatter plot using plotly. The gaps between the planes are due to a plotting problem, the input space is completely covered.

The compressed DT from the example in Figure 8.1 contains 83 nodes. It was computed with a mean computation time of 9.75s, as seen in Figure 8.3.

The number of nodes of a DT according to the equivalence description as described in Section 4.2.2, can be calculated without compression for the depth of each layer $d = \sum_{i=0}^{n-2} m_i$ with the number of filters in each layer m_i as described in Section 4.2.2 as:

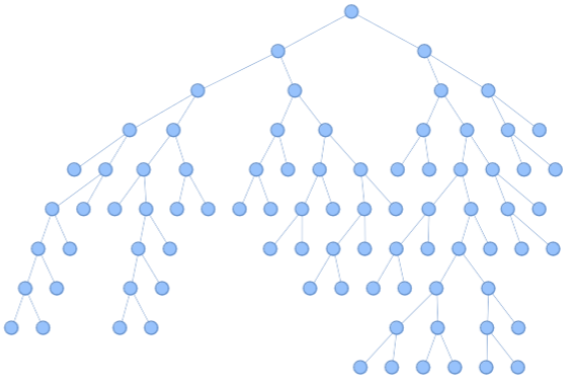


Figure 8.1: Compressed DT equivalent to the FF-DNN of an MCC controller, represented with pyvis from a networkx graph. The rules and expressions are node labels that are not visible at this zoom factor. Both models have the same output ($\delta = 1\mathrm{e}{-4}$) for a sampled grid, clearly confirming the correctness of the implementation. The corresponding input range was set as an invariant for further compression.

$$\#_{\text{nodes}} = \sum_{i=0}^{d-1} 2^i \ .$$

This formula was tested by computing the DT with the equivalence description but without compression and summing the number of nodes for different linear-ReLU FF-DNN architectures and hidden sizes.

For an architecture as in Listing 8.1, it can be computed as $d = 2 + 2hid$. For $hid = 8$, such a DT already consists of $\sum_{i=0}^{18-1} 2^i = 262143$ nodes, which corresponds to a compression ratio of 99.97 % for the number of nodes. For this size, the calculation of DT was terminated after a calculation time of 1.5h. However, for other small sizes of *hid*, it was also observed that the calculation time without compression starts to become unmanageable compared to the calculation with compression.

The results are summarized in Table 8.1 [LV23a].

Compression	$\#_{\text{nodes}}$	Computation time
☐	262143	> 1.5h
☑	83	9.75s

Table 8.1: Comparison of measurement results and calculations for the construction of a DT from the simple model without and with compression of the NN2EQCDT algorithm.

8.1.2 Discussion

The principle of equivalence description of Aytekin [Ayt22] could be verified by implementation, testing, and application to a simple model. The presented compression method seems to be a

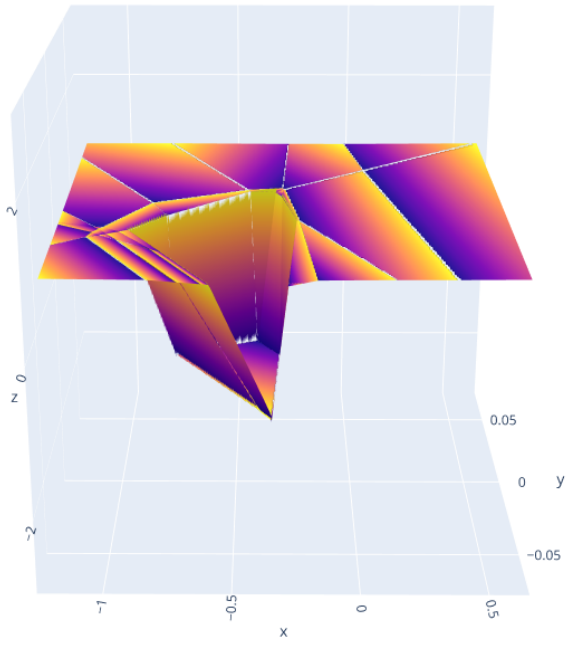


Figure 8.2: 3D visualization with the DT regions for the MCC.

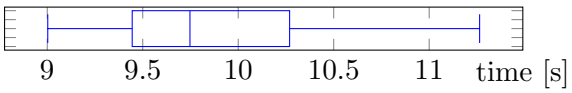


Figure 8.3: Boxplot ($n = 30$) for the computation time of constructing a DT with the NN2EQCDT algorithm for the simple model.

useful tool in the transformation to increase the explainability of FF-DNN-based DRL policies since the transformed, relatively small DT model and visualization can be used to attribute actions to observations. But in this form, it is not meaningful enough to intuit a learned strategy. Probably, for this purpose, the environment with its dynamics must be included to explain the agent's reactions and their effects on the next observations.

The transformation was successfully tested on a learned DRL model in a benchmarking environment. However, the computed compression rate of 99.97% cannot be considered representative without further evaluation. Also, a general statement about the performance of this approach for more complex environments and larger models cannot be made yet.

The NN2EQCDT algorithm can in principle transform any linear-ReLU FF-DNN models of arbitrary size for the input and output dimensions. The number of coefficients and variables in the transformed DT would then correspond to the size of the input dimension and the number of output values to the size of the output dimension, but this has not yet been implemented due to implementation difficulties but is conceptually possible. Also, only three dimensions can be easily visualized together, more dimensions require more work and probably splitting or reducing the information [LV23a].

8.2 Attack scenario

In this section, the learned agent behavior of the attack scenario from Chapter 7 is fully explained by applying the NN2EQCDT algorithm. This application will be published in [VL23].

8.2.1 Motivation

While the scientific corpus agrees that DRL-based agents are a valuable research topic for cybersecurity in CNI, their effectiveness can only be stated (1) indirectly and (2) on a case-by-case basis. Indirectly, because there is no direct method that can determine an agent's DRL policy. Publications provide analyses of rewards and simulation states, as in Section 7.3; however, it is well known that optimizing a metric, i.e., maximizing the reward, is not necessarily the same as solving the problem behind it. Second, many, if not most, publications lack long-term simulations, considering only specific, well-described, scenarios. Therefore, a DRL-based agent capability for generalization has been derived, but not fully proven [VL23].

Therefore, in this section, it is shown that the presented NN2EQCDT algorithm is generally and practically capable of transforming the derived FF-DNN of DRL-based agents to fully explain the learned behavior of the agent.

8.2.2 NN2EQCDT application

In this section, we evaluate the attack scenario described in Chapter 7. For this purpose, the learned DDPG actor models, as described in Section 3.5.3, of the attacker and defender are transformed into equivalent but compressed DTs using the NN2EQCDT algorithm as described in Chapter 6. Both agents use the same actor model FF-DNN architecture, as described in Section 8.1.1 and illustrated in Listing 8.1. To explain the decision process for different input regions, two DTs are generated for these different input regions, specified as invariants, as shown in Figure 8.4 and Figure 8.5.

Looking at the measured sensor values as shown in Figure 7.6, it can be observed that they always lie in the box $x, y \in [0.92\,\text{p.u.}, 1.02\,\text{p.u.}]$ for both buses. This can therefore be specified as an invariant for further compression when transforming the learned agent model into a DT, since domain experts may be interested in explaining the learned strategy only in this range.

This DT consists of only one node, which provides output values of < -5 for all possible combinations of input voltage values, respecting the invariants. The direct output values of a model based on the input values have no units. Only the interpretation of the output values results in units.

This constraint on the output values also applies to the DT in Figure 8.5. For this purpose, the invariants of $x, y \in [0.8\,\text{p.u.}, 1.2\,\text{p.u.}]$ were used, since they contain the SVC of possible voltage fluctuations and an additional buffer for other instabilities, as described in Section 3.1.5. Since the attacker agent has a reactive power setpoint range of $[-1.5\,\text{MVar}, 1.5\,\text{MVar}]$, all output values are clipped to the actual setpoints of $-1.5\,\text{MVar}$. This behavior was also observed in Section 7.3.

$$
\begin{aligned}
&-0.25949511x \\
&+-0.84641606y \\
&+-5.34225273 \\
&\bullet
\end{aligned}
$$

Figure 8.4: DT of the learned attacker agent. $x, y \in [0.92\,\text{p.u.}, 1.02\,\text{p.u.}]$*, i.e., only the observed bus voltage range, see Figure 7.6*

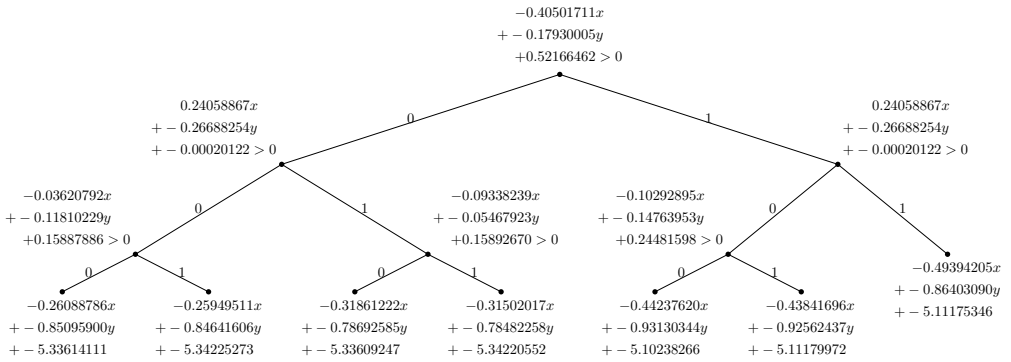


Figure 8.5: DT of the learned attacker agent only in the SVC bus voltage range

The DTs in Figure 8.4 and the one in Figure 8.5 were created using the *decision-tree* template of the *forest* package of LaTeX. They were generated from the direct output of the implementation of NN2EQCDT as networkx trees. No tool was available for this generation of DTs from networkx trees, so *NetworkXDT2LaTeXForestDT*[1] was developed for such automatic conversion.

8.2.3 Completeness validation

Die Strategie des Angreiferagenten kann nicht nur durch Inspektion des äquivalenten und komprimierten DT, der aus dem DNN-Modell des Agentenakteurs mit Hilfe des NN2EQCDT-Algorithmus transformiert wurde, erklärt werden, wie in Section 8.2.2 beschrieben, sondern es kann auch validiert werden, dass die visuell extrahierte Strategie vollständig auf der Grundlage der gemessenen Werte aus Section 7.3 extrahiert wurde.

This means that the model does not contain other substrategies that were not detected by visual inspection. The DNN model of the learned actor may, in general, contain hidden behaviors that are not extracted by visual inspection of measurements because there may be only a limited set of varying observations. For example, the DNN model of the learned actor might include other actions for observations outside the observed range of $[0.92\,\text{p.u.}, 1.02\,\text{p.u.}]$, i.e., in, $x, y \in [0.8\,\text{p.u.}, 0.92\,\text{p.u.}) \cup (1.02\,\text{p.u.}, 1.2\,\text{p.u.}]$, but which are still compatible with the SVC. However, this voltage range does not occur in the measurements, so the actor's actions in this range may differ and the model may have implemented different strategies for this

[1] The NetworkXDT2LaTeXForestDT tool can be found under: `https://gitlab.com/arl-experiments/NetworkXDT2LaTeXForestDT`

range. The equivalence transformation of the actor model used in Section 8.2.2 would identify and validate such an unknown strategy.

8.3 Conclusion and Discussion

The concept and implementation of the NN2EQCDT algorithm were evaluated to transform FF-DNN of DRL models of agent actuators directly into equivalent and compressed DTs. This was done for a model solving a benchmark problem and for a model learned with the autocurriculum in a power grid environment. In this way, the embedded strategies of the DNN models can be fully explained by visual inspection of the transformed DTs, and the extracted strategies can be validated for completeness as described in Section 8.2.3.

Although the developed XRL method worked for these scenarios, it needs to be further evaluated for more complex learned strategies in more realistic scenarios. This is particularly relevant for critical applications, as justified in Section 2.2. To learn more complex scenarios, multiple factors may need to be optimized, as described in Section 7.5. In such adapted contexts, it is necessary to evaluate the impact on the size and thus the explainability as well as the generation performance of the transformed DT. Not only does the method presented seem to help explain the simple strategies learned, but there is no known direct conceptual constraint that works for larger networks, with higher input and output dimensions, and in principle with other layers, as described in Section 4.2.2.

No further objective or subjective evaluation of the developed XRL method was performed either, as described in Section 3.6.4. However, this is necessary to increase confidence in the method itself and reliability for domain experts.

9 Conclusion and Future Work

9.1 Conclusion

9.1.1 NN2EQCDT

In this work, the NN2EQCDT algorithm was presented in Chapter 6, which can directly and accurately transform FF-DNNs into compressed DTs. The split points of such transformed DTs can also be visualized in a plot of a two-dimensional input against a one-dimensional output. Using a simple model, it was shown that a compressed DT can be significantly smaller than an uncompressed one, as described in Section 8.1. Therefore, such compressed DTs can be used to accurately trace output regions to input regions, enabling domain experts to evaluate policies for harmful behavior. In particular, they can identify unknown-unknown behaviors of the learned policy models to counter them.

This method can transform entire FF-DNNs but also allows the specification of invariants for later analysis in only certain regions to further compress the transformed DTs and thus increase its explainability. DTs also enable formal verification to help domain experts verify certain properties of the model. The transformation can be performed at any time, thus the NN2EQCDT algorithm can provide post-hoc explanations as well as explanations during training. The implementation of the algorithm works directly with learned FF-DNN models in PyTorch format, so there is no need to modify standard policy-based DRL methods to be as universal as possible.

Moreover, this method can be used to accurately analyze the learned strategies of black-box FF-DNNs through indirect interpretation by domain experts. Therefore, for use in CNI, this method has the potential to fundamentally improve explainability, user trust, and also system safety through better understanding.

9.1.2 Attack scenario

In addition, the NN2EQCDT algorithm was evaluated for its explainability capability in critical domains through an application in a power grid scenario as described in Chapter 7. For this purpose, an ARL attacker and a defender agent were used in an autocurriculum. The idea was that the attacker would learn an oscillatory strategy, but this could not be achieved. Instead, both agents simply learned opposite setpoints for reactive power injection to their injection capabilities.

The attacker's strategy could then be successfully explained using the NN2EQCDT by inspecting the same and compressed DTs. These were transformed from the FF-DNN of the agent's actor model, as shown in Section 8.2. The transformed DT could be used not only to explain the strategy but also to validate that the visually extracted strategy is complete based on the measurements of Section 7.3, i.e., that it does not contain any other sub-strategies that were not detected during the visual inspection. Such validations are necessary for critical applications since the FF-DNN actor models used must not contain unknown behaviors.

Therefore, the non-achievement of learning the oscillating strategy in the agent policy is not a major problem, because the goal was to directly and accurately explain the entire agent policy to be able to validate it in general, and not to do this specifically for more complex strategies. That this is possible in general has already been shown with the general evaluation of the NN2EQCDT algorithm in Section 8.1.

9.1.3 Answer to research question

The research question described in Section 2.3 can thus be answered, that the explainability of FF-DNN policy models in DRL methods, and in particular in the ARL approach, can be increased by the NN2EQCDT algorithm. This algorithm can directly and accurately transform the entire FF-DNN into compressed and thus understandable DT as described in Section 9.1.1. Transformed DTs can then be analyzed, the observed behavior of the policy model can be evaluated, they can be verified to check certain properties, and strategies can be indirectly derived. If done carefully, unknown unknown behaviors of the policy model can be identified or their existence can be ruled out. This minimizes the risk of agents having harmful effects through ARL and thus increases the confidence of domain experts as users in the operation of the system in power grids.

The hypothesis stated can be confirmed in that the NN2EQCDT algorithm simplifies a learned input-output mapping with its exact parameters of an FF-DNN model within its architecture by using dynamic and further compression to drastically reduce the size of the generated DT. It uses lossless compression by pruning only paths of transformed DTs that are invalid to get the exact input-output mappings of FF-DNNs. This allows accurate validation of whether the agent has learned strategies with harmful effects in subsets of input-output mappings by allowing regions to be specified as invariants that rarely occur and would not be known without such a method.

9.2 Future Work

9.2.1 NN2EQCDT

For future work, it may be interesting to further benchmark the NN2EQCDT algorithm in terms of computation time and compression ratio for different architectures and sizes of FF-DNNs learned with different policy-based DRL methods. This could involve using different targets in different environments. It could also attempt to find evidence of whether the compression ratio and computation time are representative in general, i.e., whether the number of nodes and the computation time of the NN2EQCDT do not grow exponentially on average.

The current implementation is limited in its input and output dimensions, but the NN2EQCDT concept is not. It can therefore be generalized to arbitrarily large input and output dimensions by extension. For other use cases, it could also be interesting to use other layers for precise transformations that need to be implemented and evaluated. In addition, for visualization of more than three dimensions together, multiple combinations of three dimensions or other reduction methods such as Principle Component Analysis (PCA) could be evaluated.

Furthermore, it could be tried to encode transformed DTs back into parameters of FF-DNNs to get smaller, possibly minimized FF-DNNs. This could be used for further training since it is better regularized compared to DTs. If DTs are modified beforehand, for example by eliminating harmful behavior and encoding expert rules directly, the back-transformed FF-DNNs could potentially be trained more efficiently.

To further increase the explainability of large DTs, similar adjacent regions can be grouped so that split points in them are removed for further compression. The similarity threshold and application regions in DTs can be defined so that these introduced approximations to the original policy can be accurately calculated and incorporated into risk management.

Furthermore, with the transformed DTs features and extreme values could be analyzed, because not only the input-output mapping but also other metrics such as which features are more important and where possible extreme values lie can be of interest for explanations. For example, large slops of expressions in leaf nodes are indicators of harmful behavior, which is not obvious because this method cannot be used to make statements about the next input, i.e., the next observations.

One approach to overcome this limitation could be to combine a DT of an agent's policy with a transformed DT of the relevant world model. The resulting graph could then also be attempted to be further compressed. Since the output of such a combination would also be the input for the next iteration, the graph would contain loops that could be further analyzed. Further, an attempt could be made to identify fixed areas that the system would loop through forever, thus keeping the power system in a healthy state without violating any constraints that might otherwise lead to a blackout. With such a combination, it would then be interesting to first identify healthy areas that then successively converge to unhealthy areas in the action observation loop to identify actions against them.

9.2.2 Power grid scenarios

In addition, in the context of the power grid scenario, an attempt could be made, for example, to replicate the oscillating strategy as the attacker-agent strategy by having the agents take turns in performing their actions as described in Section 7.5. An attempt could also be made to introduce self-play for this purpose. In general, an attempt could also be made to develop more sophisticated agent strategies, which could then be tested for explainability using the NN2EQCDT algorithm.

9.2.3 General

The transformed DTs allow domain experts to analyze the exact input-output mappings, i.e., what agents learned, but not why they learned certain strategies and why they may have avoided others. Therefore, it may be interesting to combine the NN2EQCDT algorithm with other exact policy-based self-explainable methods such as IBMDP or PRL as described in Section 4.1.2 to summarize or generalize policies to better explain potentially learned strategies.

Furthermore, Ternary Vector Lists (TVLs) could be tried to use to get an exact, summary of the rules. The rules, i.e., the checks at split-point nodes of a DT, are Boolean functions in that they evaluate to either true or false. Therefore, they can be represented by a Binary Vector List (BVL). To reduce the large number of such vectors, TVLs can be used. It reduces the exponential expansion of BVLs by reducing the number of ternary vectors, with *orthogonal blocking*. Thereby two vectors, which differ only in one position, are compressed into one [SD00].

Another idea is to use the NN2EQCDT algorithm in combination with general XAI methods such as SHAP and general-purpose methods such as LIME, as described in Section 3.6.2.2.

10 Tool usage

The following external tools were used for this thesis. This does not include the self-written tools like the NN2EQCDT implementation and the NetworkXDT2LaTeXForestDT conversion tool.

1. DeepL translator (`https://www.deepl.com`, not DeepL Write): Used to improve the quality of self-written texts by translating self-written English texts into German, improving them, and translating them back into English.
2. Grammarly (`https://app.grammarly.com`, not writing assistance or GrammarlyGo): Used to check grammar
3. PyCharm (`https://www.jetbrains.com/pycharm/`): Used for coding in python.
4. Relevant python frameworks and libraries (described in text)
   (a) PyTorch
   (b) OpenAI Gym, or rather the new Gymnasium from Farama Foundation (`https://gymnasium.farama.org`)
   (c) graphviz
   (d) matplotlib
   (e) networkx
   (f) numpy
   (g) sympy
   (h) z3-solver
   (i) numexpr
   (j) plotly
   (k) DDPG implementation of spinningup
   (l) palaestrAI framework
   (m) MIDAS framework
   (n) pandapower
   (o) jupyter lab
   (p) docker

I declare that no generative text AI tools, such as ChatGPT, DeepL Writer, or the GrammarlyGo or its writing assistance, were used to generate whole passages of text.

Figures

Literature

[APE21] Dilini Almeida, Jagadeesh Pasupuleti, and Janaka Ekanayake. "Comparison of reactive power control techniques for solar PV inverters to mitigate voltage rise in low-voltage grids". In: *Electronics* 10.13 (2021), p. 1569.

[Arr+20] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information fusion* 58 (2020), pp. 82–115.

[Axe86] Robert Axelrod. "An evolutionary approach to norms". In: *American political science review* 80.4 (1986), pp. 1095–1111.

[Ayt22] Caglar Aytekin. *Neural Networks are Decision Trees*. 2022. arXiv: 2210.05189 [cs.LG].

[Bak+20] Bowen Baker et al. *Emergent Tool Use From Multi-Agent Autocurricula*. 2020. arXiv: 1909.07528 [cs.LG].

[Bal23] Stephan Balduin. *MIDAS Commercial Data Simulator*. [retrieved: 07, 2023]. 2023. URL: https://midas-mosaik.gitlab.io/midas/modules/comdata.html.

[BC14] Jimmy Ba and Rich Caruana. "Do deep nets really need to be deep?" In: *Advances in Neural Information Processing Systems* 27 (2014), pp. 2654–2662.

[BDE22] BDEW. *Redispatch 2.0*. [retrieved: 07.2023]. 2022. URL: https://www.bdew.de/energie/redispatch-20/.

[BPS18] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. "Verifiable reinforcement learning via policy extraction". In: *Advances in Neural Information Processing Systems* 31 (2018).

[Bun09] Bundesministerium der Justiz. *Gesetz über das Bundesamt für Sicherheit in der Informationstechnik (BSI-Gesetz - BSIG)*. [retrieved: 07.2023]. Aug. 2009. URL: https://www.gesetze-im-internet.de/bsig_2009/BJNR282110009.html.

[Bun21] Bundesministerium der Justiz. *Gesetz über das Bundesamt für Sicherheit in der Informationstechnik (BSI-Gesetz - BSIG)*. [retrieved: 07.2023]. June 2021. URL: https://www.bgbl.de/xaver/bgbl/start.xav?startbk=Bundesanzeiger_BGBl&jumpTo=bgbl121s1122.pdf#__bgbl__%2F%2F*%5B%40attr_id%3D%27bgbl121s1122.pdf%27%5D__1689340201934.

[BW89a] Mesut Baran and Felix F Wu. "Optimal sizing of capacitors placed on a radial distribution system". In: *IEEE Transactions on Power Delivery* 4.1 (1989), pp. 735–743.

[BW89b] Mesut E Baran and Felix F Wu. "Network reconfiguration in distribution systems for loss reduction and load balancing". In: *IEEE Transactions on Power Delivery* 4.2 (1989), pp. 1401–1407.

[BW89c] Mesut E Baran and Felix F Wu. "Optimal capacitor placement on radial distribution systems". In: *IEEE Transactions on Power Delivery* 4.1 (1989), pp. 725–734.

[Coo23] Stephen A Cook. "The complexity of theorem-proving procedures". In: *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*. 2023, pp. 143–152.

[DB08] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14*. Springer. 2008, pp. 337–340.

[DB11] Leonardo De Moura and Nikolaj Bjørner. "Satisfiability Modulo Theories: Introduction and Applications". In: *Communications of the ACM* 54.9 (2011), pp. 69–77.

[Del21] Deloitte. *Cyber-Sicherheit und Cyber Resilienz für die Schweizer Stromversorgung*. Tech. rep. [retrieved: 07.2023]. Schweizerische Eidgenossenschaft - Bundesamt für Energie BFE, June 2021. URL: https://pubdb.bfe.admin.ch/de/publication/download/10524.

[DK17] Finale Doshi-Velez and Been Kim. *Towards A Rigorous Science of Interpretable Machine Learning*. 2017. arXiv: 1702.08608 [stat.ML].

[DK79] Richard Dawkins and John Richard Krebs. "Arms races between and within species". In: *Proc. R. Soc. Lond. B* 205.1161 (1979), pp. 489–511.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. "A machine program for theorem-proving". In: *Communications of the ACM* 5.7 (1962), pp. 394–397.

[EGH16] Vasileios A Evangelopoulos, Pavlos S Georgilakis, and Nikos D Hatziargyriou. "Optimal operation of smart distribution networks: A review of models, methods and future research". In: *Electric Power Systems Research* 140 (2016), pp. 95–106.

[Ele10] Electric Power Research Institute. *Standard Language Protocols for Photovoltaics and Storage Grid Integration - Developing a Common Method for Communicating with Inverter-Based Systems*. [retrieved: 07, 2023]. May 2010. URL: http://assets.fiercemarkets.net/public/smartgridnews/1020906LangProtocolsPVStorageGridIntegrate.pdf.

[Eur23a] European comission. *2030 Climate Target Plan*. [retrieved: 07, 2023]. July 2023. URL: https://climate.ec.europa.eu/eu-action/european-green-deal/2030-climate-target-plan_en.

[Eur23b] European comission. *A clean energy transition*. [retrieved: 07, 2023]. July 2023. URL: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal/energy-and-green-deal_en.

[Eur23c] European comission. *Cause of climate change*. [retrieved: 07, 2023]. July 2023. URL: https://climate.ec.europa.eu/climate-change/causes-climate-change_en.

[Eur23d] European comission. *Consequences of climate change*. [retrieved: 07, 2023]. July 2023. URL: https://ec.europa.eu/clima/climate-change/consequences-climate-change_en.

[Eur23e] European comission. *Paris Agreement*. [retrieved: 07, 2023]. July 2023. URL: https://climate.ec.europa.eu/eu-action/international-action-climate-change/climate-negotiations/paris-agreement_en.

[FCD15] John K Feser, Swarat Chaudhuri, and Isil Dillig. "Synthesizing data structure transformations from input-output examples". In: *ACM SIGPLAN Notices* 50.6 (2015), pp. 229–239.

[FHM18] Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI].

[Fis+18] Lars Fischer et al. *Adversarial Resilience Learning - Towards Systemic Vulnerability Analysis for Large and Complex Systems*. 2018. arXiv: 1811.06447 [cs.AI].

[Fou23a] Farama Foundation. *Mountain Car Continuous*. [retrieved: 07.2023]. 2023. URL: https://gymnasium.farama.org/environments/classic_control/mountain_car_continuous/.

[Fou23b] PyTorch Foundation. *PyTorch Linear*. [retrieved: 07.2023]. 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html.

[Fra23] Fraunhofer IEE and University of Kassel. *Pandapower 2.0 Cigre benchmark power grid implementation*. [retrieved: 06, 2023]. 2023. URL: https://pandapower.readthedocs.io/en/v2.0.0/networks/cigre.html.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. [retrieved: 07.2023]. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[GS94] John J Grainger and WUliam D Stevenson Jr. *Power system analysis*. McGraw-Hill series in electrical and computer engineering, 1994.

[Haa+18] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: 1801.01290 [cs.LG].

[Har68] Garrett Hardin. "The tragedy of the commons". In: *Science* 162.3859 (1968), pp. 1243–1248.

[HCD21] Alexandre Heuillet, Fabien Couthouis, and Natalia Diaz-Rodriguez. "Explainability in deep reinforcement learning". In: *Knowledge-Based Systems* 214 (2021), p. 106685.

[Hec89] Douglas D Heckathorn. "Collective action and the second-order free-rider problem". In: *Rationality and Society* 1.1 (1989), pp. 78–100.

[HGS15] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: 1509.06461 [cs.LG].

[Hof+19] Robert R. Hoffman et al. *Metrics for Explainable AI: Challenges and Prospects*. 2019. arXiv: 1812.04608 [cs.AI].

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[Hug+18] Edward Hughes et al. "Inequity aversion improves cooperation in intertemporal social dilemmas". In: *Advances in Neural Information Processing System* (2018), pp. 3330–3340.

[Inv22] Investopedia. *Black Swan in the Stock Market: What Is It, With Examples and History*. [retrieved: 07.2023]. June 2022. URL: https://www.investopedia.com/terms/b/blackswan.asp.

[Jar+09] Kevin Jarrett et al. "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 2146–2153.

[JL18] Peizhong Ju and Xiaojun Lin. "Adversarial Attacks to Distributed Voltage Control in Power Distribution Networks with DERs". In: *Proceedings of the Ninth International Conference on Future Energy Systems*. ACM. 2018, pp. 291–302. ISBN: 9781450357678. DOI: 10.1145/3208903.3208912.

[JVW20] T. Jaunet, R. Vuillemot, and C. Wolf. "DRLViz: Understanding Decisions and Memory in Deep Reinforcement Learning". In: *Computer Graphics Forum* 39.3 (2020), pp. 49–61. DOI: 10.1111/cgf.13962. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13962.

[KHM14] R Kabiri, DG Holmes, and BP McGrath. "The influence of pv inverter reactive power injection on grid voltage regulation". In: *2014 IEEE 5th International Symposium on Power Electronics for Distributed Generation Systems (PEDG)*. IEEE. 2014, pp. 1–8.

[Kin76] James C. King. "Symbolic Execution and Program Testing". In: *Commun. ACM* 19.7 (July 1976), pp. 385–394. ISSN: 0001-0782. DOI: 10.1145/360248.360252.

[Kro] Dirk P. Kroese. *Cross-Entropy Method*. [retrieved: 07.2023]. Brisbane 4072, Australia. URL: https://people.smp.uq.edu.au/DirkKroese/ps/eormsCE.pdf.

[Lan+17] Marc Lanctot et al. "A unified game-theoretic approach to multiagent reinforcement learning". In: *Advances in Neural Information Processing Systems* (2017).

[Lap20] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more*. Packt Publishing Ltd, 2020.

[LB08] Ellen Liu and Jovan Bebic. *Distribution system voltage performance analysis for high-penetration photovoltaics*. Tech. rep. National Renewable Energy Lab (NREL), Golden, CO (United States), 2008.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[Leh22] Sebastian Lehnhoff. *Lecture Smart Grid Management*. 2022.

[Lei+17] Joel Z. Leibo et al. "Multi-agent Reinforcement Learning in Sequential Social Dilemmas". In: *CoRR* abs/1702.03037 (2017). arXiv: 1702.03037.

[Lei+19] Joel Z. Leibo et al. *Autocurricula and the Emergence of Innovation from Social Interaction: A Manifesto for Multi-Agent Intelligence Research*. 2019. arXiv: 1903.00742 [cs.AI].

[Les+18] Timothée Lesort et al. "State representation learning for control: An overview". In: *Neural Networks* 108 (2018), pp. 379–392.

[Liu+08] Y. Liu et al. "Distribution System Voltage Performance Analysis for High-Penetration PV". In: *2008 IEEE Energy 2030 Conference*. 2008, pp. 1–8. DOI: 10.1109/ENERGY.2008.4781069.

[LL17] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Advances in neural information processing systems* 30 (2017).

[Log22] Torben Logemann. "How Digital Twins, Cybersecurity and Trustworthiness in Smart Grid contribute to the European Climate Goals 2030". [unpublished, used in Smart Grid Research lecture]. Sept. 2022.

[LV23a] Torben Logemann and Eric MSP Veith. "NN2EQCDT: Equivalent Transformation of Feed-Forward Neural Networks as DRL Policies into Compressed Decision Trees". In: vol. 15. [retrieved: 07, 2023]. IARIA. ThinkMind, June 2023, pp. 94–100. ISBN: 978-1-68558-046-9. URL: https://www.thinkmind.org/index.php?view=article&articleid=cognitive_2023_1_160_40107.

[LV23b] Torben Logemann and Eric MSP Veith. *Simple Voltage Attack Explainability repository*. [retrieved: 07, 2023]. 2023. URL: https://gitlab.com/arl-experiments/simple-voltage-attack-explainability.

[Mar04] Henryk Markiewicz. *Standard EN 50160 – Voltage Characteristics of Public Distribution Systems*. [retrieved: 07.2023]. July 2004. URL: https://powerquality.blog/2021/07/22/standard-en-50160-voltage-characteristics-of-public-distribution-systems/.

[Mni+13] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[Moo90] Andrew William Moore. *Efficient Memory-based Learning for Robot Control*. Tech. rep. University of Cambridge, 1990.

[MSP+20] Eric MSP Veith et al. "Analyzing Power Grid, ICT, and Market Without Domain Knowledge Using Distributed Artificial Intelligence". In: *arXiv e-prints* (2020), arXiv–2006.

[NH10] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 807–814.

[NKA21] Duy T. Nguyen, Kathryn E. Kasmarik, and Hussein A. Abbass. *Towards Interpretable ANNs: An Exact Transformation to Multi-Class Multivariate Decision Trees*. 2021. arXiv: 2003.04675 [cs.LG].

[Nob57] Clyde E. Noble. "Human Trial-and-Error Learning". In: *Psychological Reports* 3.2 (1957), pp. 377–398. DOI: 10.2466/pr0.1957.3.h.377. eprint: https://doi.org/10.2466/pr0.1957.3.h.377.

[NR21] Thanh Thi Nguyen and Vijay Janapa Reddi. “Deep reinforcement learning for cyber security”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[NRC20] Muddasar Naeem, Syed Tahir Hussain Rizvi, and Antonio Coronato. “A gentle introduction to reinforcement learning and its application in different fields”. In: *IEEE access* 8 (2020), pp. 209320–209344.

[Ope23] OpenAI Spinning Up. *Part 2: Kinds of RL Algorithms*. [retrived: 07.2023]. July 2023. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[Ost00] Elinor Ostrom. “Collective action and the evolution of social norms”. In: *Journal of economic perspective* 14.3 (2000), pp. 137–158.

[Ost90] Elinor Ostrom. “Governing the Commons: The Evolution of Institutions for Collective Action”. In: *Cambridge University Press* (1990).

[Per+17] Julien Perolat et al. “A multi-agent reinforcement learning model of common-pool resource appropriation”. In: *Advances in Neural Information Processing Systems* (2017), pp. 3643–3652.

[PV20a] Erika Puiutta and Eric Veith. “Explainable reinforcement learning: A survey”. In: *International cross-domain conference for machine learning and knowledge extraction*. Springer. 2020, pp. 77–95.

[PV20b] Erika Puiutta and Eric M. S. P. Veith. “Explainable Reinforcement Learning: A Survey”. In: *Machine Learning and Knowledge Extraction. CD-MAKE 2020*. Dublin, Ireland: Springer, Cham, 2020, pp. 77–95. DOI: 10.1007/978-3-030-57321-8_5.

[Qin+22] Yunpeng Qing et al. *A Survey on Explainable Reinforcement Learning: Concepts, Algorithms, Challenges*. 2022. arXiv: 2211.06665 [cs.LG].

[Rap74] Anatol Rapoport. “Prisoner’s dilemma - recollections and observations”. In: *Game Theory as a Theory of a Conflict Resolution* (1974), pp. 17–34.

[RB14] Ranga V Ramasesh and Tyson R Browning. “A conceptual framework for tackling knowable unknown unknowns in project management”. In: *Journal of Operations Management* 32.4 (2014), pp. 190–204.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why should i trust you?” Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[Sal+17] Ahmad EL Sallab et al. “Deep Reinforcement Learning framework for Autonomous Driving”. In: *Electronic Imaging* 29.19 (Jan. 2017), pp. 70–76. DOI: 10.2352/issn.2470-1173.2017.19.avm-023.

[Sam+13] Spyridon Samothrakis et al. “Coevolving gameplaying agents: Measuring performance and intransitivities”. In: *IEEE Transactions on Evolutionary Computation* 17.2 (2013), pp. 213–226.

[Sch+17] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[Sch+20] Julian Schrittwieser et al. "Mastering Atari, Go, chess and shogi by planning with a learned model". In: *Nature* 588.7839 (Dec. 2020), pp. 604–609. DOI: 10.1038/s41586-020-03051-4.

[Sch12] Adolf J Schwab. *Elektroenergiesysteme: Erzeugung, Transport, Übertragung und Verteilung elektrischer Energie*. Springer-Verlag, 2012.

[Sch73] Thomas C Schelling. "Hockey helmets, concealed weapons, and daylight saving: A study of binary choices with externalities". In: *Journal of Conflict resolution* 17.3 (1973), pp. 381–428.

[SD00] Bernd Steinbach and Chr Dorotska. "Orthogonal Block Building Using Ordered Lists of Ternary Vectors". In: *Boolean Problems, Proceedings of the 4th International Workshops on Boolean Problems*. Vol. 21. Citeseer. 2000, p. 22.

[Sil+17] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI].

[Sta+16] William R Stauffer et al. "Components and characteristics of the dopamine reward utility signal". In: *Journal of Comparative Neurology* 524.8 (2016), pp. 1699–1711.

[Tas14] Task Force C6.04.02. "Benchmark systems for network integration of renewable and distributed energy resources". In: *Elektra - CIGRE's digital magazine* 575 (2014).

[Top+21] Nicholay Topin et al. "Iterative bounding mdps: Learning interpretable policies via non-interpretable methods". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 11. 2021, pp. 9923–9931.

[Tur+11] Konstantin Turitsyn et al. "Options for control of reactive power by distributed photovoltaic generators". In: *Proceedings of the IEEE* 99.6 (2011), pp. 1063–1073.

[US 23] U.S. Department of Energy. *Open Energy Data Initiative*. [retrieved: 07, 2023]. 2023. URL: https://data.openei.org/about.

[VDE17] Forum Netztechnik/Netzbetrieb im VDE (FNN) VDE. *Zusammenfassung des Entwurfs VDE-AR-N 4110:2017-02*. Tech. rep. [retrieved: 07.2023]. Feb. 2017. URL: https://ivrenergy.de/wp-content/uploads/2018/11/tar-ms-zusammenfassung-de-data-compressed.pdf.

[VDE19] VDE. *Zellulares Energiesystem - Ein Beitrag zur Konkretisierung des zellularen Ansatzes mit Handlungsempfehlungen*. [retrieved: 07.2023]. Frankfurt am Main, May 2019. URL: https://www.vde.com/resource/blob/1884494/98f96973fcdba70777654d0f40c179e5/studie---zellulares-energiesystem-data.pdf.

[Vei+22] Eric MSP Veith et al. *Learning to Attack Powergrids with DERs*. 2022. arXiv: 2204.11352 [cs.CR].

[Vei22] Dr.-Ing. Eric MSP Veith. *ARL Adversarial Resilience Learning - Förderung von Ideennachwuchs Forschungsvorhaben von KI-Nachwuchsgruppen*. 2022.

[Vei23] Eric MSP Veith. "An Architecture for Reliable Learning Agents in Power Grids". In: *ENERGY 2023 Editors* (2023), p. 19.

[VL23] Eric MSP Veith and Torben Logemann. “Towards Explainable Attacker-Defender Autocurricula in Critical Infrastructures”. [not published yet]. 2023.

[VWU23] Eric M. S. P. Veith, Arlena Wellßow, and Mathias Uslar. “Learning new attack vectors from misuse cases with deep reinforcement learning”. In: *Frontiers in Energy Research* 11 (2023). ISSN: 2296-598X. DOI: 10.3389/fenrg.2023.1138446.

[Wol+23] Thomas Wolgast et al. *ANALYSE – Learning to Attack Cyber-Physical Energy Systems With Intelligent Agents*. 2023. arXiv: 2305.09476 [cs.CR].

[Wol96] David H Wolpert. “The lack of a priori distinctions between learning algorithms”. In: *Neural computation* 8.7 (1996), pp. 1341–1390.

[Yam88] Toshio Yamagishi. “Seriousness of social dilemmas and the provision of a sanctioning system”. In: *Social psychology quarterly* (1988), pp. 32–42.

[ZBM16] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. “Graying the black box: Understanding DQNs”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 1899–1908. URL: https://proceedings.mlr.press/v48/zahavy16.html.

[ZL16] H Zhu and H. J. Liu. “Fast local voltage control under limited reactive power: Optimality and stability analysis”. In: *IEEE Trans. Power Syst.* 31 (2016), pp. 3794–3803.

Index