Analyzing Exact Output Regions of Reinforcement Learning Policy Neural Networks for High-Dimensional Input-Output Spaces

Torben Logemann[a] and Eric MSP Veith[b]
Carl von Ossietzky University Oldenburg,
Research Group Adversarial Resilience Learning,
Oldenburg, Germany

Keywords: Decision Tree, Reinforcement Learning, Explainability, Neural Network.

Abstract: Agent systems based on deep reinforcement learning have achieved remarkable success in recent years. They have also been applied to a variety of research topics in the field of power grids, as such agents promise real resilience. However, deep reinforcement learning agents cannot guarantee behavior, as the mapping of the entire input space to the output of even a simple feed-forward neural network cannot be accurately explained. For critical infrastructures, such black box models are not acceptable. To ensure an optimized trade-off between learning performance and explainability, this paper relies on efficient regularizable feed-forward neural networks and presents an extension of the algorithm NN2EQCDT to transform the networks into pruned decision trees with significantly fewer nodes to be accurately explained. In this paper, we present a methodological approach to further analyze the decision trees for high-dimensional input-output spaces and analyze an agent for a power grid experiment.

1 INTRODUCTION

Deep Reinforcement Learning (DRL)—the notion of agents that learn from interacting with their environment—is at the core of many of these remarkable successes, beginning with its breakthrough in 2013 by end-to-end learning of Atari games (Mnih et al., 2013) and Double Deep Q-Learning (DDQN) (Van Hasselt et al., 2016) and culminating in developments such as AlphaGo (Zero), AlphaZero (Silver et al., 2018) and MuZero (Schrittwieser et al., 2020). Since the 2013 hallmark paper by Mnih et al., researchers have improved training performance, sensibility to hyperparameters, exploration, or sample efficiency through algorithms such as Twin-Delayed Deep Deterministic Policy Gradient (DDPG) (TD3) (Fujimoto et al., 2018), Proximal Policy Optimization (PPO) (Schulman et al., 2017), and Soft Actor Critic (SAC) (Haarnoja et al., 2018). The current research corpus shows that these agents have proven that they are capable of handling complex tasks.

DRL agents promise true resilience by learning to counter the unknown unknowns. However, unlike intrinsically interpretable DRL models (Puiutta and Veith, 2020), no guarantees can yet be made about the behavior of DRL agents learned with black-box models. This is, however, a necessity for operators, since no responsibility can be taken for an unknown control system that cannot be validated, especially when it is used in such critical or very critical areas as Critical National Infrastructures (CNIs).

Agents deployed in complex environments, such as complex networked systems, are potentially confronted with many different situations and learn complex behaviors to fulfill their goals. For example, in (Veith et al., 2023; Veith et al., 2024) Adversarial Resilience Learning (ARL) attack agents are trained to cause voltage band violations in a power grid. This goal is achieved by exploiting a weakness in the use of voltage regulators in the grid in use.

To gain a better insight into how the control strategies work compared to the classical reward and action implications on the victim busses, the NN2EQCDT algorithm was developed (Logemann and Veith, 2023), which exactly transforms Deep Neural Network (DNN) networks into pruned Decision Trees (DTs). This allows not only individual trajectories to be explained, but also the mapping of entire observation regions to functions for the actions on the basis of all input points from the region (Veith

[a] https://orcid.org/0000-0002-2673-397X
[b] https://orcid.org/0000-0003-2487-7475

Logemann, T. and Veith, E. M.
Analyzing Exact Output Regions of Reinforcement Learning Policy Neural Networks for High-Dimensional Input-Output Spaces.
DOI: 10.5220/0012928000003886
In *Proceedings of the 1st International Conference on Explainable AI for Neural and Symbolic Methods (EXPLAINS 2024)*, pages 96-107
ISBN: 978-989-758-720-7

and Logemann, 2023; Logemann, 2023). It is embedded in the ARL architecture in (Veith, 2023), which aims to create a framework for an agent that can learn sample-efficiently, but also provide insights into the behavior of the agents.

For more complex control strategies with higher input/output dimensions, it becomes more difficult to give guarantees, since such spaces make it considerably more difficult to read the control strategies directly from and visualize the transformed DTs. Furthermore, it is not easily possible to provide explanations and guarantees for control agents over a longer time horizon. This is relevant, e.g., in the case of agents for voltage maintenance, as voltage band violations must never occur.

In this paper, we present a methodological approach that can represent the properties of regions in higher dimensional input-output spaces in relation to each other. The input space is completely disjointly divided into polytopes by a DT generated by the NN2EQCDT algorithm. For a better understanding of the regions, the polytopes are also approximated by inner boxes and their neighborhood relation is represented in the new presented concept of the Neighborhood Graph (NHG), which works for arbitrarily high-dimensional regions. The edges can represent properties between these regions. In this way, (abrupt) transitions in the control planes of the agent can be identified. We illustrate our approach in a well-known gymnasium environment, which is accessible for both visual inspection and our approach, to show the feasibility of our method.

The remainder of this paper is structured as follows: First, related work is presented in section 2 and an overview of the whole system is given in section 3. Then the construction of the equivalent pruned DT from a Feed-Forward Deep Neural Network (FF-DNN) is generally described in section 4. The exact representation of the output region is also described in section 5. In addition, the calculation of the inner boxes in section 6 and the handling of observation space constraints in section 7 are described. With these calculations, an example of a controller trained in the MountainCarContinuous-v0 environment (MCC) is given in section 8. Then the NHG is presented in section 9 and consistently validated with the MCC example. After that, a higher input and output dimensional experiment in the power grid domain is analyzed. Finally, a discussion in section 11 and references to future work follow.

2 RELATED WORK

2.1 Deep Reinforcement Learning

Reinforcement Learning (RL) is the process of learning through interaction. When an RL agent interacts with its environment and thereby observing the consequences of its actions in form of rewards, it can learn to alter its own behavior in response to the received rewards. RL has the paradigm of trial-and-error-learning and is influenced by the optimal control field as well as balancing the trade-off between exploration and exploitation. It is based on the Markov Decision Process (MDP), which is a quintruplet $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, that describes that an agent observes the state $s_t \in \mathcal{S}$ from its environment at time t and takes the action $a_t \in \mathcal{A}$ as a response to the discounted (γ the discount factor) reward $r_t \in \mathcal{R}$. If the state is reset after each episode, then the sequence of states, actions and rewards is a trajectory of the policy. The return of a trajectory is given by the discounted accumulation of rewards $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$ This transitions the state to the next state $s_{t+1} \in S$ with the probability $p_t \in \mathcal{T}$. If the system is only partially observable, a Partially-Observable Markov Decision Process (POMDP) further specifies Ω as the set of observations and O as the conditional observation properties, $\Omega \times O \subseteq S$. In general, reinforcement learning aims to learn a policy, $a_t \sim \pi(\cdot|s_t)$. Finding the optimal policy π^*, that maximizes the expected return from all states,

$$\pi^* = \arg\max_{\pi} \mathbb{E}[R|\pi] \,, \tag{1}$$

is the optimization problem of all reinforcement learning algorithms (Arulkumaran et al., 2017).

2.2 Explainability for Deep Reinforcement Learning

DNNs are inherently opaque, as the meaning of any particular set of nodes eludes a human. Therefore, DRL policies are similar black boxes (Jaunet et al., 2020); since experts cannot readily understand why a particular action was taken especially in the context of the overall learned policy, the lack of transparency limits trust in DRL agents (Qing et al., 2022). Recent papers survey the landscape of the very active field of research of eXplainable Reinforcement Learning (XRL) (Arrieta et al., 2020; Puiutta and Veith, 2020), to which we refer the interested reader. We will focus on equivalent representations of policy networks as decision trees.

2.2.1 Explaining with Decision Trees

Transforming the agent's policy into a DT is intuitively justified, as we can easily accept decisions in a basis of if-then-else conditions; DTs seem to be easy to interpret (Du et al., 2019). Algorithms such as Viper (Bastani et al., 2018) create the DT from samples of the original policy. But when the agents' state/action space and its strategies become more complex, the DT also grows in size, making its creation and even evaluation slow. For this reason, approaches such as distilling into Soft Decision Trees (SDTs) (Irsoy et al., 2012; Frosst and Hinton, 2017) have been proposed (Coppens et al., 2019), because SDTs are more space efficient.

However, such distillation methods require the agent produce trajectories the DT can be constructed from (imitation learning), which have the drawback that the DT can only cover what the agent has done so far; there is no guarantee that the DT covers the agent's behavior in general.

To this end, another approach has been proposed in which the policy network itself is converted to an equivalent DT (Aytekin, 2022). But a problem with this approach was that the resulting DT can grow very large. We have previously extended the algorithm to prune unreachable nodes off from the DT dynamically while creation and, therefore, minimize it (Logemann, 2023).

3 SYSTEM OVERVIEW

In fig. 1 an overview about the whole system is visualized. First, the NN2EQCDT algorithm is used to transform a given DNN exactly into a truncated DT. The regions that the DT splits form convex polytopes. These are used as a matrix inequality system for Linear Programming (LP) approaches to compute various properties and the neighborhoods that are compacted into a NHG. General constraints, such as global outer boxes, are included in the calculations with the NN2EQCDT algorithm as well as those with LP. The NHG is a tool that can be used for further analysis of region transitions and properties.

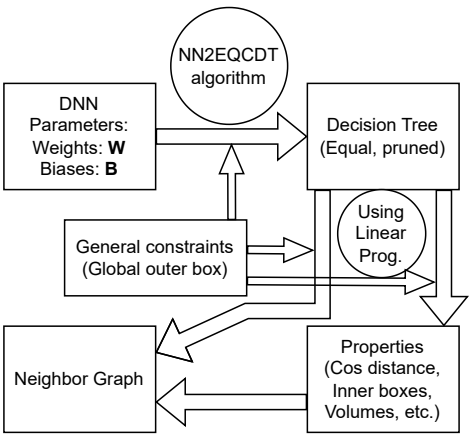


Figure 1: System overview of the interplay between the NN2EQCDT algorithm and the NHG calculation.

4 THE NN2EQCDT ALGORITHM

In this section, we briefly describe the original algorithm NN2EQCDT (cf. algorithm 1). For further details and evaluations, please see to the original publication (Logemann, 2023).

Data: ANN weight matrix $\boldsymbol{W}$, bias matrix $\boldsymbol{B}$, general constraints c_g
Result: Pruned Decision Tree T
begin
  $\hat{\boldsymbol{W}} = \boldsymbol{W}_0$
  $\hat{\boldsymbol{B}} = \boldsymbol{B}_0^\top$
  $rules$ = CALC_RULE_TERMS$\left(\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}\right)$
  T, new_SAT_leaves = CREATE_INITIAL_SUBTREE$(rules, c_g)$
  ADD_SAT_PATHS(T, new_SAT_leaves)
  SET_HAT_ON_SAT_NODES $\left(T, new\_SAT\_leaves, \hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}\right)$
  for $i = 1, \ldots, n-1$ **do**
    SAT_paths = POP_SAT_PATHS(T)
    for SAT_path in SAT_paths **do**
      $\boldsymbol{a}$ = COMPUTE_A_ALONG(SAT_path)
      SAT_leave = LAST_ELEMENT(SAT_path)
      $\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}$ = GET_LAST_HAT_OF_LEAVE(T, SAT_leave)
      $\hat{\boldsymbol{W}} = (\boldsymbol{W}_i \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{W}}$
      $\hat{\boldsymbol{B}} = (\boldsymbol{W}_i \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{B}} + \boldsymbol{B}_i^\top$
      $rules$ = CALC_RULE_TERMS$\left(\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}\right)$
      new_SAT_leaves = ADD_SUBTREE $\left(T, SAT\_leave, rules, c_g\right)$
      SET_ON_NODES $\left(T, new\_SAT\_leaves, \hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}\right)$
      ADD_SAT_PATHS(T, new_SAT_leaves)
    end
  end
  CONVERT_FINAL_RULE_TO_EXPR(T)
  PRUNE_TREE(T)
end

Algorithm 1: Main loop of the NN2EQCDT algorithm.

The weight and bias matrices $\boldsymbol{W}_i$ and $\boldsymbol{B}_i$ of layer i from the FF-DNN model are processed layer by layer. Initial rules are calculated from the effective matrices

(function CALC_RULE_TERMS). The initial effective matrices are simply the weight and bias matrices of the first layer. A rule is the symbolic representation for checking the input data on a node in DT. The evaluation of a rule with input data decides which further path the input data takes in DT.

A rule can be imagined as an evaluation of the application of the ReLU function to the current input data, which is calculated for all possible input data with the effective matrices. The jth rule of the ith layer thus sums the matrix coefficients of the jth row multiplied by their respective input variables, so:

$$(\text{rule}_i)_k \mathrel{\hat{=}} \sum_j (\hat{\boldsymbol{W}}_i)_{k,j} x_j + (\hat{\boldsymbol{B}}_i)_{k,j} > 0 \,. \tag{2}$$

The DT is built from top to bottom, whereby the k-th rule is assigned to all nodes of the $\mathit{offset} + k$-th line of the DT. The offset is the index of the last layer of the already built DT. Before adding each node n with a rule to DT, a check is made to see whether the rule from node n together with all the rules from the nodes in the path above it from the root node to node n and the general boundary constraints can be satisfied. This can be done either with Satisfiability Modulo Theories (SMT) solvers or in this case by checking with LP as described in the next sections. If the rules together are unsatisfiable, i.e., there can be no input, so that the evaluation of DT that takes this path, the node n and thus further subtrees are not added to keep the size of DT dynamically small.

All further subtrees that are calculated are only appended to the satisfying (SAT) leaf nodes of the already created DT. Therefore, SAT leaf nodes must be saved, which is why they are returned after a subtree has been created (see return values of the functions CREATE_INITIAL_SUBTREE or ADD_SUBTREE).

The calculation of $\hat{\boldsymbol{W}}$ and $\hat{\boldsymbol{B}}$ depends on the slope vector $\boldsymbol{a}$, which represents the "decisions" of the input data $\boldsymbol{x}$ by the rules on the path from the root node to a SAT control node. Thus, $\boldsymbol{a}_k = 1$ if $\text{rule}_{\text{node}_k}(\boldsymbol{x}) \equiv \mathit{true}$ otherwise $\boldsymbol{a}_k = 0$. Since the calculation of the rules in turn depends on $\hat{\boldsymbol{W}}$ and $\hat{\boldsymbol{B}}$, each subtree that is generated from rules with the slope vector of a SAT leave node $n_{\text{SAT\_leave}}$ is appended to that leave node.

So after the initial subtree has been created from the first set of rules using the function CREATE_INITIAL_SUBTREE, the rules are not assigned to all nodes in a row in DT, but only to the nodes within a subtree. Starting with the second layer, all further layers are iterated through, and for each, the path from the root to the SAT leaf node (function POP_SAT_PATHS) is iterated to first calculate the corresponding slope vector (function COMPUTE_A_ALONG). This is used to calculate $\hat{\boldsymbol{W}}$ and $\hat{\boldsymbol{B}}$ together with the current $\boldsymbol{W}_i$ and $\boldsymbol{B}_i$. After calculating the rules from the effective matrices, these are used together with the general constraints for the SAT check to create a subtree that is added to the SAT leaf node (function LAST_ELEMENT) of the given SAT path.

In order to access the latest $\hat{\boldsymbol{W}}$ and $\hat{\boldsymbol{B}}$ for the calculation of the effective weight matrices in the next layer iteration, they are stored in the new SAT leaf nodes. In order to be able to iterate over the new SAT paths, they are calculated and added to the tree from the new SAT leaves (function ADD_SAT_PATHS).

Finally, after iterating all layers, the rules of the last SAT leaves are converted into expressions, and the DT can be further pruned by removing nodes with unnecessary rules if they are evaluated equally for all possible inputs.

5 EXACT OUTPUT REGION REPRESENTATION

The nodes of a DT transformed with the NN2EQCDT algorithm contain n rules on a direct path from the root to a leaf node with expression (exclusive), which are linear inequalities of the form:

$$\text{rule}_{0 \le k \le (n-1)} \mathrel{\hat{=}} \sum_j (a_k)_j x_j + b_k > 0 \,. \tag{3}$$

The rules represent half-space constraints, as each one subdivides the entire input space. Each node of a path of a DT from the root to a leaf node effectively further subdivides an already segmented space with a then bounded half-space. All leaf nodes therefore lie in output regions that are described by the intersection of the half-spaces of their respective paths above. These output regions are therefore convex polytopes of the standard form (Vanderbei et al., 2014) $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}\}$ where

$$\boldsymbol{A} = \big((a_k)_j\big) \quad \text{and} \quad \boldsymbol{b} = \big(b_k\big) \tag{4}$$

are the concatenation of the coefficients of a rule in the columns for all rules in the rows.

For a proof see section 12.

Note that the strict inequalities of the comparison against zero, which originate from the ReLU resolution, must be reformulated into the standard form. This requires the formulation of a non-strict inequality, which is why the case of equality must be checked separately.

Whether linear inequalities have a common solution can be determined by solving "fake" LP problems of the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & 0 \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b} \,. \end{aligned} \tag{5}$$

Solving LPs problems is comparatively more efficient and this can be used for dynamic path checking instead of checking the constraints with general SMT solvers as originally used in (Logemann and Veith, 2023).

Solving such LP problems can also be used to check whether points or other convex polytopes lie in or intersect the convex polytope in question by adding further representative constraints to the LP problem, such as:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & 0 \\ \text{subject to} \quad & \boldsymbol{A}\,\boldsymbol{x} \leq \boldsymbol{b}, \\ & \boldsymbol{x} = \boldsymbol{y}, \qquad \text{for point } \boldsymbol{y} \qquad (6a) \\ \text{or} \quad & \boldsymbol{A}'\boldsymbol{x} \leq \boldsymbol{b}', \qquad \text{for polytope } \mathcal{P}' \,. \qquad (6b) \end{aligned}$$

Each leaf node l of a DT represents the linear equation of the hyperplane bounded by the respective polytope $\mathcal{P}_l$. This denotes the actual function that is applied in the region to derive the equivalent output from a FF-DNN, such as the setpoint of an agent actor from its policy network.

6 INNER BOXES

The exact representation of the polytopes as an intersection of the half-spaces becomes more difficult to imagine for higher dimensions. For a general overview, however, smaller but definitive intervals for all dimensions are more suitable, as in such a representation the boundaries for each dimension are independent of the others. This is achieved by inner boxes with maximum volume of the polytopes.

According to Proposition 1 of (Bemporad et al., 2004), an inner box with maximum volume of a full-dimensional convex polytope $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, can be obtained by optimizing the following LP problem:

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{y}} \quad & \sum_{j \in D} \ln y_j \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{A}^+\boldsymbol{y} \leq \boldsymbol{b}\,, \end{aligned} \qquad (7)$$

where $\boldsymbol{A}^+$ is the positive part of $\boldsymbol{A}$, i. e., $a_{ij}^+ = \max(0, a_{ij})$.

The optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ denotes the inner box with maximum volume $\mathcal{B}(\boldsymbol{x}^*, \boldsymbol{x}^* + \boldsymbol{y}^*)$. A box is represented as $\mathcal{B}(\boldsymbol{l}, \boldsymbol{u}) = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\}$, where $\boldsymbol{l}$ and $\boldsymbol{u}$ are real d-vectors.

The convexity of the polytopes is ensured by construction with the half-space splitting in the DTs of by NN2EQCDT algorithm, as described in section 5. A

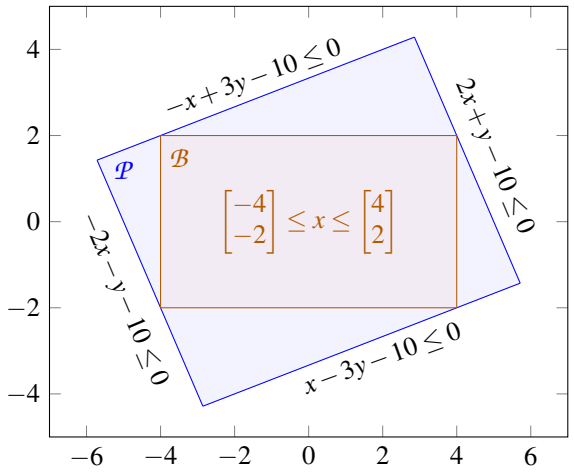


Figure 2: Example for the volume-maximized inner box $\mathcal{B}$ (orange) of a convex polytope $\mathcal{P}$ (blue).

polytope $\mathcal{P}$ on the other hand is full-dimensional if it has an interior point. An interior point of $\mathcal{P}$ is a point $\hat{\boldsymbol{x}} \in \mathbb{R}^d$ that satisfies $\boldsymbol{A}\hat{\boldsymbol{x}} < \boldsymbol{b}$. This cannot be checked directly by LP optimization as it does not work with strict inequalities. But it can be checked using the following LP problem:

$$\begin{aligned} \max_{\boldsymbol{x},x_0} \quad & x_0 \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \mathbf{1}x_0 \leq \boldsymbol{b}, \\ & x_0 \leq 1\,. \end{aligned} \qquad (8)$$

If this is solved successfully, with $\boldsymbol{x}^*$ being an optimal solution and x_0^* being the optimal value, then $\boldsymbol{x}^*$ is the inner point of $\mathcal{P}$, if $x_0^* > 0$, $\mathcal{P}$ is full-dimensional. In the case of $x_0^* < 0$, $\mathcal{P}$ is empty and in the case of $x_0^* = 0$, $\mathcal{P}$ is neither full-dimensional nor empty (Fukuda, 2015).

In fig. 2 an example of an inner box with maximum volume of a polytope is shown. Although the inner box has a maximum volume for the interior of the polytope, the regions $\mathcal{P} \setminus \mathcal{B}$ are significant, i. e. $\mathcal{B}$ comprises only

$$\frac{\text{vol}(\mathcal{B})}{\text{vol}(\mathcal{P})} = \frac{32}{53.88} = 59.4\%$$

of the volume of $\mathcal{P}$.

However, the approximation may be sufficient for an overview, since the high-dimensional regions are easier to visualize with the box intervals than with the more complicated inequality constraints of the respective polytope.

7 OBSERVATION SPACE CONSTRAINTS

Normally, the observation space has natural or artificial value limits in each dimension, so that only values within known intervals can be entered. These can be used as a global constraint for the NN2EQCDT algorithm (see fig. 1) to generate smaller DT by pruning nodes or subtrees where the input data falls entirely within unreachable regions. The calculation of the inner box can also be modified to respect these intervals by

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{y}} \quad & \sum_{j\in D} \ln y_j \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x}+\boldsymbol{A}^{+}\boldsymbol{y} \leq \boldsymbol{b}\,, \\ & \boldsymbol{o}_{min} \leq \boldsymbol{x} \leq \boldsymbol{o}_{max}\,, \\ & \boldsymbol{o}_{min} \leq \boldsymbol{x}+\boldsymbol{y} \leq \boldsymbol{o}_{max} \end{aligned} \tag{9}$$

with $(\boldsymbol{o}_{min}, \boldsymbol{o}_{max})$ being the bounds for the input or observation space. Note that they span the outer box $\mathcal{B}_{out}(\boldsymbol{o}_{min}, \boldsymbol{o}_{max})$ because $\boldsymbol{x}$ and $\boldsymbol{x}+\boldsymbol{y}$ are both points constrained by the respective polytope and globally on these constraints together, in which the maximization of an inner box is performed. Therefore, the edges of all i-th inner boxes $\mathcal{B}_i(\boldsymbol{x}_i, \boldsymbol{x}_i+\boldsymbol{y}_i)$ can be constrained directly by the bounds of the outer box without affecting the volume maximization. Since the boundary conditions of the outer box are basically also additional half-space intersection points, the intersection set $\mathcal{B}_{out} \cap \mathcal{P} = \mathcal{P}'$ is also a convex polytope. In addition to convexity, the calculation of the inner box also requires that the polytope is full-dimensional, which in turn can be checked with the optimization of the LP problem described in section 6. Therefore, the resulting polytope must have the standard form given by

$$\mathcal{P}' = \left\{ \boldsymbol{x} \in \mathbb{R}^d \;\middle|\; \boldsymbol{A}' = \begin{pmatrix} \boldsymbol{A} \\ I \\ -I \end{pmatrix} \boldsymbol{x} \leq \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{o}_{max} \\ -\boldsymbol{o}_{min} \end{pmatrix} = \boldsymbol{b}' \right\}. \tag{10}$$

8 MOUNTAIN CAR EXAMPLE

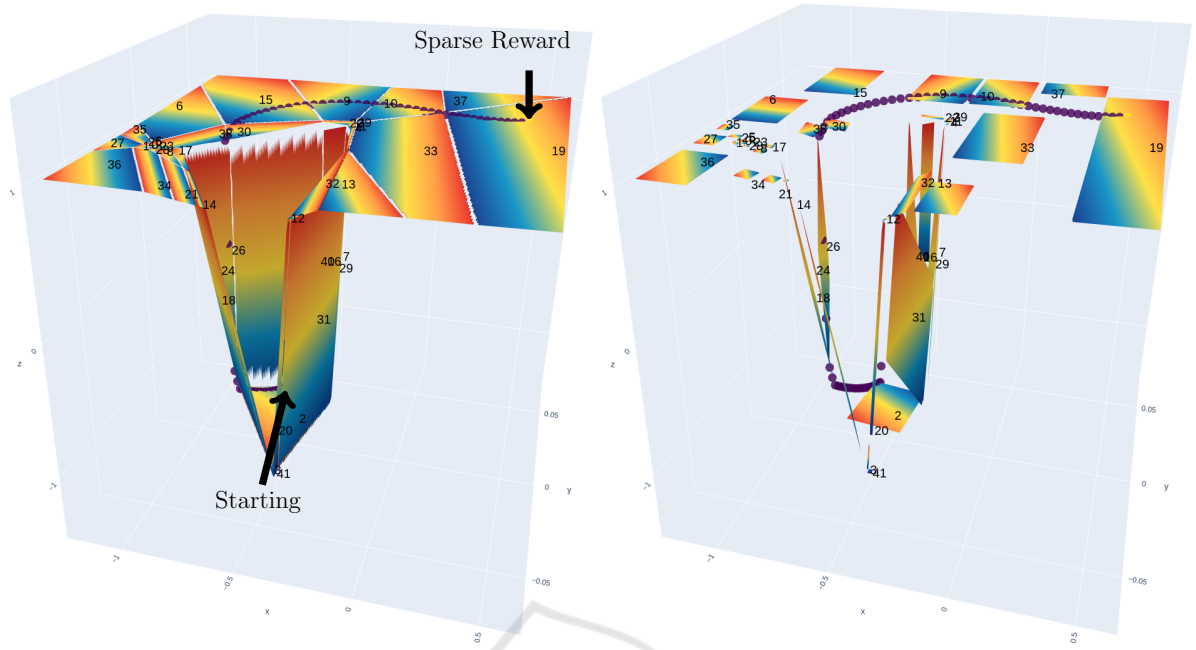
In fig. 3 the output regions of the same trained model are shown for a control strategy of the MCC as used in (Logemann and Veith, 2023). The MCC has two inputs in the observation space, the first being the position of the vehicle along the x-axis and the second being the velocity of the vehicle, and an actuator that controls the directional force exerted on the vehicle.

An agent was trained that maximizes the reward (there is only a sparse reward for reaching the mountain). Its DNN was transformed into a DT using the NN2EQCDT algorithm. In fig. 3a the output regions of this DT are plotted according to their exact polytopes, while in fig. 3b the output regions are represented by the inner boxes of the polytopes. The output regions are plotted against the linear actuator functions specified in the leaf nodes of the respective DT.

Using the Gymnasium interface, which provides space types such as box intervals, the observation space can be constrained as described in section 7. In the case of MCC, such intervals are given by $-1.2 \leq x \leq 0.6$ and $-0.07 \leq y \leq 0.07$, which is why they are used for the modified optimization problem.

In addition to the output regions, the actual action points the simulation runs through are displayed. Action points are defined here as the tuple of the observations (here: x, y), their mapping of the neural network to the actuator value (here: $(x,y) \mapsto z$) and the reward value (here: $(\texttt{env}, s, x, y, z) \mapsto r$, with `env` being the environment and s the state of the environment (from which the possible partial observations obtained), and on which the subsequent actuator value is applied to). So here the action points are represented by $a = ((x,y),(z),r)$. In fig. 3, they are plotted as 3D points by the mapping $((x,y) \mapsto z)$. The trace of action points starts on the lower plane and ends on the higher planes. The rewards are represented by the color of the points. Here, one can see that there is only one high spare reward (yellow) at the end, that corresponds to the car reaching the mountain.

In addition to the output regions, the actual action points that are fed into the simulation are displayed. Action points are defined here as the tuple of the observation values (here: x, y), their mapping of the neural network to the actuator value (here: $(x,y) \mapsto z$) and the reward value (here: $(\texttt{env}, s, x, y, z) \mapsto r$. `env` is the environment and s is the state of the environment from which the possible partial observations result to which the subsequent actuator value is applied. The action points are therefore represented here by $a = ((x,y),(z),r)$. In fig. 3 they are drawn as 3D points through the mapping $((x,y) \mapsto z)$. The track of action points starts on the lower level and ends on the higher levels. The rewards are represented by the color of the points. Here you can see that there is only one high sparse reward (yellow) at the end, which corresponds to the car reaching the mountain.



(a) Exact output regions with polytopes

(b) Output regions represented by inner boxes of polytopes

Figure 3: Output regions of a FF-DNN model trained in the MCC with the global constraints and actual action points.

9 NEIGHBORHOOD GRAPH

For three dimensions, the output regions can be drawn and observed as polytopes; for higher dimensions, the approximating, but dimension-independent, representation of inner boxes can be used for better visualization. Only linear functions are applied to these output regions. In order to better understand the interaction of output regions and linear functions of an agent in higher-dimensional input and output spaces, the concept of a NHG is introduced.

A NHG is a simple, undirected, weighted, cyclic and finite graph $G = (V, E)$. The vertices $v \in V$ represent initial regions. Two vertices (v, u) are connected by an edge $e = (v, u) \in E$ iff. the initial regions $\mathcal{P}_v$ and $\mathcal{P}_u$ represented by v and u are direct neighbors, which is given by $I = \mathcal{P}_v \cap \mathcal{P}_u \neq \emptyset$. If they are neighbors, the intersection contains the neighboring edge points that both have in common, due to the relaxation of the strict inequalities as described in section 5. The intersection I can again be checked for non-emptiness by solving the LP problem from eq. (6b) and checking whether it has a solution.

For two hyperplanes described by the linear functions $v \mapsto \sum a_i x_i + c$ and $u \mapsto \sum b_i y_i + d$ with the coefficient vectors $\boldsymbol{a} = [(a_i), c]^\top$ and $\boldsymbol{b} = [(b_i), d]^\top$ and which are evaluated with input points in the polytopes, the cosine similarity is calculated by

$$\cos\alpha = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\| \|\boldsymbol{b}\|} . \tag{11}$$

If we consider closed-loop systems, such as the model of section 8 as a controller for the dynamics of the system of MCC, all model outputs for each input $\boldsymbol{x}_i$ control the system so that the next input point $\boldsymbol{x}_{i+1}$ lies in the ε sphere of the previous input. This means that $d(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) < \varepsilon$, where the Euclidean distance is $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|$ for a global, minimal ε.

As can be seen in the example, the ε can be comparatively small so that the points of the action point trace cross the polytopes through their neighbors. However, if two hyperplanes of neighboring polytopes are not similar, there is a sudden change in control behavior, which can cause a sudden change in the state and behavior of the system. The non-similarity of the hyperplanes of two neighboring polytopes can therefore be an important measure. For this reason, the modified cosine distance $\cos_{\text{distance}} = 1 - |\cos\alpha|$ is indicated by the size of the edges in the NHG, as shown in the example in fig. 4. It is modified by taking the absolute value of the cosine similarity, as the direction of the normal vectors of the hyperplane is irrelevant.

The magenta-colored edges in fig. 4 represent the trace of the action points of the MCC example. The weight of the edges (thickness) describes how often

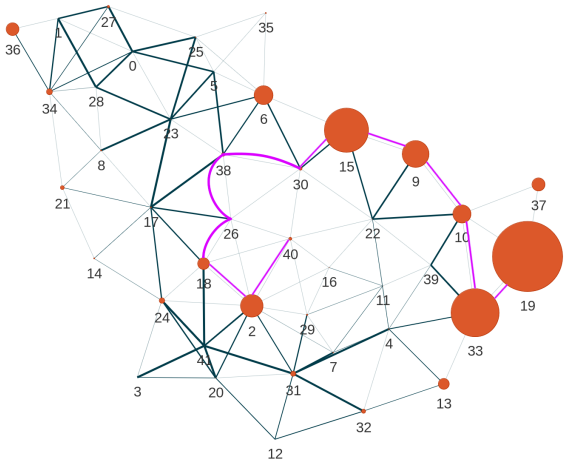


Figure 4: NHG for the FF-DNN model trained in MCC with the global constraints and on the one hand the cosine distance as dark blue edges and the action point trace as magenta edges.

an edge was crossed by two consecutive action points in relation to each other. It can be observed that the action point trace starts at the vertex 40 and runs over vertex 2, which is represented by the lower hyperplane in fig. 3. Furthermore, the trace runs across vertex 26, which is a steep hyperplane, and finally the trace ends on the upper levels until the agent gets the reward in vertex 19, which has a large size.

The size of the polytopes in the output region can also be significant, as the loop system can remain in a state where the inputs fall into the same output region for a long time and the action points are sampled as they pass through the same hyperplane function. The size of the polytopes is measured by their volume, which is visualized by the size of the vertices in the NHG. The volume is calculated with the library of (TuLiP control, 2024). The library can also be used to calculate polytope properties such as the center and radius of the Chebyshev sphere and the bounding box. In addition to these dimensions, the boundaries and the volume of the inner boxes of the polytopes, as described in section 6, and the expression of the hyperplane are mapped to the vertices in the NHG.

10 EXPERIMENT

In the following scenario (Logemann, 2024) the CIGRE Medium Voltage (MV) benchmark power grid (Task Force C6.04.02, 2014), as shown in fig. 5, is used.

Several PV systems are connected to the busses as generators and various loads with time-dependent profiles. The power fed in by the PV systems and

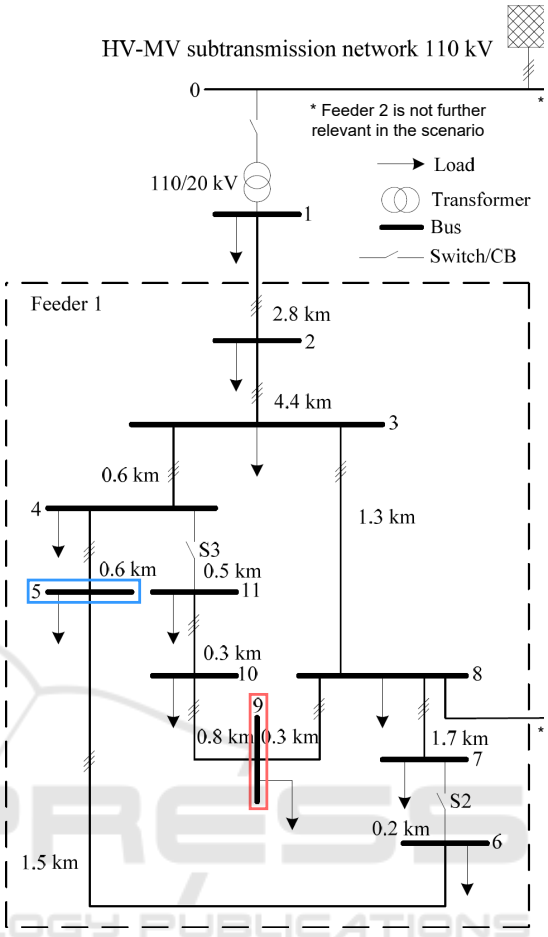


Figure 5: CIGRE MV power grid, adapted from (Fraunhofer IEE and University of Kassel, 2023); blue: bus with the Photovoltaic (PV) inverters for which the agent controls the setpoints for Active Power (AP) and Reactive Power (RP), and bus with the highest weighted Voltage Magnitude (VM) for the RP setpoint, red: bus with the highest weighted VM for the AP setpoint.

thus the VMs of the buses are time-dependent on the solar radiation caused by the simulated weather. The simulated VMs of the busses are shown in fig. 6. Between the time steps 2000 and 2500, the VMs initially increase and finally decrease due to violations of the VM constraints of the busses.

10.1 Classic Analysis

To keep the voltage within the constraints, a very simple controller agent with DRL is learned. The simulated VMs of this scenario with the controller are shown in fig. 5.

The objective of the agent in this scenario is to keep the VMs of the busses (observations: vm_b, input for the DNN of the agent) in feeder 1 as close as

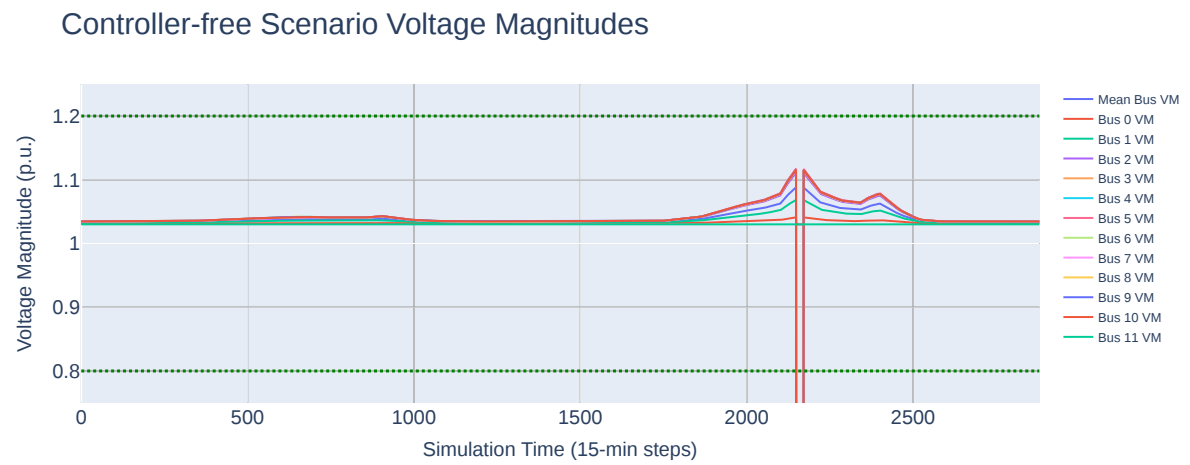


Figure 6: Time series of the VMs for the scenario without a controller agent. Between 2000 and 2500, the voltage constraints (VM must be in $[0.8, 1.2]$) are violated and the bus is switched off so that the VM drops to zero.

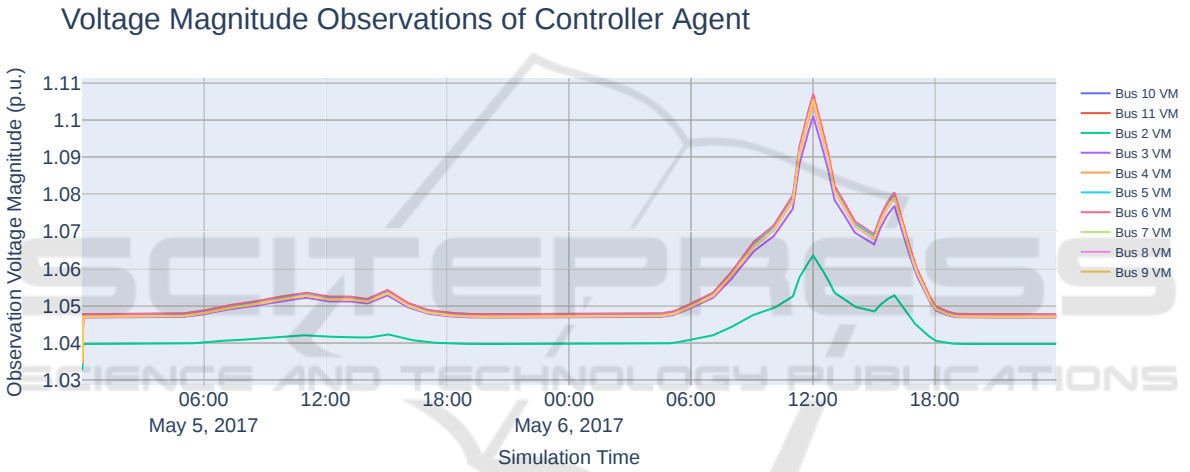


Figure 7: Time series of the VMs for the scenario without a controller agent.

possible to 1 p.u. (nominal voltage). The voltage is not only generally controlled by the AP, but is also directly dependent on the RP fed in. The balanced feed-in of AP and RP generated by a PV can be set via setpoints for the inverter (Turitsyn et al., 2011). In the scenario, the agent controls such on bus 5 (actuators) using the output of the agent's DNN, where these output values are first transformed before they are actually applied as setpoints, as shown for the simulation of the scenario in fig. 8.

To obtain the applied setpoints y_A, the function

$$y_A = \min(\max(a_{s,A} \tanh(o_A) + a_{b,A}, \{p_{\min}, q_{\min}\}), \{p_{\max}, q_{\max}\}) \quad (12)$$

with $A = \{\text{ap}, \text{rp}\}$ is first applied with output o_A of the DNN for the actuators, the action scaling factor $a_{s,A}$ and bias $a_{b,A}$ as well as the respective maximal and minimal AP setpoint ($p_{\{\min,\max\}}$) and RP setpoint ($q_{\{\min,\max\}}$) for the clipping. The $y_{p,q}$ values are then balanced to the actual setpoints, that the inverter can deliver at the current time via the P-Q characteristic (Turitsyn et al., 2011).

10.2 Exact Controller Analysis

The DNN of the controller agent was transformed into an exact DT, which in turn was used to create the NHG as described in section 3. It consists of only two nodes, with almost all simulated data points falling into the observation region of one node. Thus, almost all observed VMs are evaluated by a set of two linear expressions of the form

$$o_A = \sum_i w_{A,i} \text{vm}_{A,i} + c_A \ , \quad (13)$$

where $w_{A,i}$ is the weight for the VM of the ith bus $\text{vm}_{A,i}$ and c_A is the bias value for the two actuators.

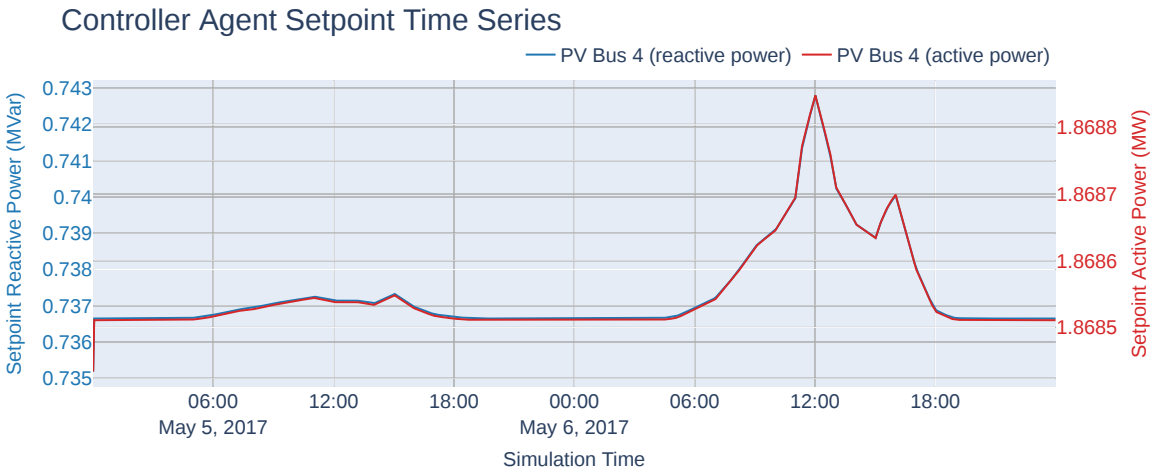


Figure 8: Time series for the setpoints of the controller agent for the PV system on bus 4; AP and RP setpoints overlap all the time.

Table 1: Weights for the observed VMs for the region in which almost all test simulation data fall.

AP actuator		**RP actuator**	
Bus b_{ap}	Weight w_{ap}	Bus b_{rp}	Weight w_{rp}
9	0.013	5	0.019
4	0.009	2	0.018
10	0.003	11	0.018
3	0.002	3	0.008
6	0.001	8	0.007
11	−0.004	7	−0.001
7	−0.004	9	−0.006
5	−0.005	4	−0.010
8	−0.006	6	−0.010
2	−0.006	10	−0.013

The weights of two different buses $b_{i,j}$ for the AP and RP actuators listed in table 1 are only approximately proportional to each other ($w_{A,i} \propto\!\!\!\sim w_{A,j}$), as the non-linear y function and the inverter balancing, as described in section 10.1, are still applied to them.

The bus with the highest weighted VM for the AP actuator is 9 and for the RP it is bus 5. Considering only the RP actuator, the learned strategy is analogous to a simple RP controller, which also controls only one RP setpoint for a PV inverter on the same bus, only from which it observes the VM (Ju and Lin, 2018).

Although, or perhaps despite this, the VM profiles of all busses in fig. 7 are very similar except for the scaling, the agent has learned properties of the topology of the network: the controller has learned to identify its own bus (5) in terms of the generally known simple RP controller strategy. Also noteworthy is the second highest weighted VM of bus 2, which has a fundamentally stronger influence on all VMs of the feeder's busses, as it is closer to the common coupling point (the 110/20 kV transformer on bus 1 in fig. 5).

11 DICUSSION

Although, the inner boxes are volume-maximized, are based on the exact polytope representation of the output regions, and can thus be used for an exact overview, they are significantly smaller than the polytopes in terms of their volume. The representation for an overview of a polytope could be extended by the inner boxes of the sub-polytopes in $\mathcal{P} \cap \mathcal{B}_{\text{in}}$ of a polytope $\mathcal{P}$ and its inner box $\mathcal{B}_{\text{in}}$. On the other hand, this would increase the complexity of the overview because then, there are multiple inner boxes for the same polytope and thus the same output region. So it has to be further evaluated in more practical, real-world scenarios if, and in which form, this inner box representation(s) can be useful for an overview.

The NHG has been successfully used to visualize the relation of the hyperplanes and other measures to each other. The volume of the polytopes and the cosine distance as well as the mapping of the other properties of the hyperplanes can be used to investigate the higher dimensional position of the linear functions of neighboring polytopes to each other. It has been shown that restricting the input space to the box spanning the action points leads to interpretable graphs in a simple real world experiment. However, the methods need to be evaluated for larger observation spaces and more input and output dimensions.

12 CONCLUSION AND FUTURE WORK

This paper presents a methodological approach to analyze the exact output regions of policy FF-DNNs for high-dimensional input-output spaces. Such an exact analysis is necessary for the understanding of agents' strategies, especially with regards to CNI operation. The paper builds upon the exact transformation of FF-DNNs into a pruned DT. The paths from a root to the leaf nodes of such an DT make up output regions that can be represented by polytopes. For an overview of a polytope, the inner box is computed. Furthermore, the output regions are depicted by vertices in the introduced NHG. They are connected by edges if they are direct neighbors to each other. Further properties, like the polytope constraints, inner box bounds and their volume, as well as, the Chebyshev ball center and radius, the bounding box, and the cosine distance of the normal vectors of the hyperplanes are also mapped to the vertices and thus hyperplanes in the output regions. Thereby, the Neighbor Graph especially allows us to visualize the relations between higher-dimensional output regions.

In future work, these analysis tools could be evaluated in real scenarios with even higher dimensions. They could also be used to analyze the learned control strategies of ARL agents in the power grid for larger input domains. This would make it possible to get a better overview of the entire possible behavior of the agent.

ACKNOWLEDGEMENTS

This work was funded by the German Federal Ministry for Education and Research (BMBF) under Grant No. 01IS22071.

REFERENCES

Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Aytekin, Ç. (2022). Neural networks are decision trees. *CoRR*, abs/2210.05189:1–8. [retrieved: 05, 2023].

Bastani, O., Pu, Y., and Solar-Lezama, A. (2018). Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31:2499–2509.

Bemporad, A., Filippi, C., and Torrisi, F. D. (2004). Inner and outer approximations of polytopes using boxes. *Computational Geometry*, 27(2):151–178.

Coppens, Y., Efthymiadis, K., Lenaerts, T., and Nowé, A. (2019). Distilling deep reinforcement learning policies in soft decision trees. In *International Joint Conference on Artificial Intelligence*.

Du, M., Liu, N., and Hu, X. (2019). Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77.

Fraunhofer IEE and University of Kassel (2023). Pandapower 2.0 cigre benchmark power grid implementation. [retrieved: 06, 2023].

Frosst, N. and Hinton, G. (2017). Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR.

Fukuda, K. (2015). Lecture: Polyhedral computation, spring 2015. [retrieved: 06.2024].

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290. [retrieved: 05, 2023].

Irsoy, O., Yildiz, O. T., and Alpaydin, E. (2012). Soft decision trees. In *International Conference on Pattern Recognition*.

Jaunet, T., Vuillemot, R., and Wolf, C. (2020). Drlviz: Understanding decisions and memory in deep reinforcement learning. *Computer Graphics Forum*, 39(3):49–61.

Ju, P. and Lin, X. (2018). Adversarial attacks to distributed voltage control in power distribution networks with DERs. In *Proceedings of the Ninth International Conference on Future Energy Systems*, pages 291–302. ACM.

Logemann, T. (2023). Explainability of power grid attack strategies learned by deep reinforcement learning agents.

Logemann, T. (2024). Power grid experiment. https://gitlab.com/arl-experiments/explains-simple-voltage-controller. [retrieved: 09, 2024].

Logemann, T. and Veith, E. M. (2023). Nn2eqcdt: Equivalent transformation of feed-forward neural networks as drl policies into compressed decision trees. volume 15, page 94–100. IARIA, ThinkMind. [retrieved: 07, 2023].

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A.

(2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602:1–9. [retrieved: 05, 2023].

Puiutta, E. and Veith, E. M. S. P. (2020). Explainable reinforcement learning: A survey. In *Machine Learning and Knowledge Extraction. CD-MAKE 2020*, volume 12279, pages 77–95, Dublin, Ireland. Springer, Cham.

Qing, Y., Liu, S., Song, J., and Song, M. (2022). A survey on explainable reinforcement learning: Concepts, algorithms, challenges. *CoRR*, abs/2211.06665:1–25. [retrieved: 05, 2023].

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347. [retrieved: 05, 2023].

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

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

TuLiP control (2024). Polytope implementation. [retrived: 06, 2024].

Turitsyn, K., Sulc, P., Backhaus, S., and Chertkov, M. (2011). Options for control of reactive power by distributed photovoltaic generators. *Proceedings of the IEEE*, 99(6):1063–1073.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, volume 30, pages 2094–2100. AAAI Press.

Vanderbei, R. J. et al. (2014). *Linear programming*. Springer.

Veith, E. M. (2023). An architecture for reliable learning agents in power grids. *ENERGY 2023 : The Thirteenth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pages 13–16. [retrieved: 05, 2023].

Veith, E. M. and Logemann, T. (2023). Towards explainable attacker-defender autocurricula in critical infrastructures. *CYBER 2023 : The Eighth International Conference on Cyber-Technologies and Cyber-Systems //*, pages 27–31. [retrieved: 06, 2024].

Veith, E. M., Logemann, T., Wellßow, A., and Balduin, S. (2024). Play with me: Towards explaining the benefits of autocurriculum training of learning agents. In *2024 IEEE PES Innovative Smart Grid Technologies Europe (ISGT EUROPE)*, pages 1–5, Dubrovnik, Croatia. IEEE.

Veith, E. M. S. P., Wellßow, A., and Uslar, M. (2023). Learning new attack vectors from misuse cases with deep reinforcement learning. *Frontiers in Energy Research*, 11:01–23. [retrieved: 05, 2023].

APPENDIX

Convexity of Intersection of Half-Spaces

Theorem 1. *Let $H_{1\leq i\leq m} \subseteq \mathbb{R}^n$ be half spaces, defined by linear inequalities, such that $H = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}^\top \boldsymbol{x} \geq b\}$, $\boldsymbol{a} \in \mathbb{R}^n$, $b \in \mathbb{R}$. Then its intersection set $A = \bigcap_{i=1}^{m} H_i$ is convex.*

Proof. First it is proofed, that half spaces are convex sets, then that the intersection of any collection of convex sets is also convex.

(1.) Show that half spaces are convex:

$$\forall \boldsymbol{x}_{1,2} \in H \subseteq \mathbb{R}^n, \forall \lambda \in [0,1] : \lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2 \in H \ .$$

Let $x_1, x_2 \in H$ and $\lambda \in [0,1]$, then

$$\begin{aligned} & \boldsymbol{a}^\top \boldsymbol{x}_1 - b \geq 0 \\ \xRightarrow{\text{Convex def.}} & \underbrace{\lambda(\boldsymbol{a}^\top \boldsymbol{x}_1 - b)}_{\geq 0} + \overbrace{\underbrace{(1-\lambda)}_{\geq 0}}^{\in [0,1]}(\boldsymbol{a}^\top \boldsymbol{x}_2 - b) \geq 0 \\ \Leftrightarrow & \lambda \boldsymbol{a}^\top \boldsymbol{x}_1 - \lambda b + \boldsymbol{a}^\top \boldsymbol{x}_2 - b - \lambda \boldsymbol{a}^\top \boldsymbol{x}_2 + \lambda b \geq 0 \\ & \boldsymbol{a}^\top (\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2) - b \geq 0 \end{aligned}$$

from this follows $\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2 \in H, \forall \boldsymbol{x}_{1,2} \in H, \lambda \in [0,1]$

(2.) Show that the intersection set of convex subsets $\mathcal{C}$ a real or complex vector space V is convex:

$$\mathfrak{C} = \bigcap_{\substack{C \subseteq V, \\ C \text{ convex}}} C \qquad \text{is convex .}$$

Let $\boldsymbol{x}_{1,2} \in \mathfrak{C}$ be two elements from the intersection set, then they are in all intersected sets: $\forall C \in \mathcal{C} : \boldsymbol{x}_{1,2} \in C$ and because of the convexity, it holds:

$$\forall \lambda \in [0,1] : \lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2 \in C$$

Because this holds for every C and thus $\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2$ are in C for all $\lambda \in [0,1]$, these elements are also in the intersection set $\mathfrak{C}$. Hence it is also convex.

(3.) From (1.) and (2.), it directly follows that the intersection set of half spaces is convex. □