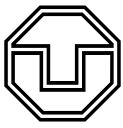

TECHNISCHE UNIVERSITÄT DRESDEN

"Friedrich List" Faculty of Transport and Traffic Sciences, Chair of Econometrics and Statistics, esp. in the Transport Sector

Master Thesis

Explainability in Deep Reinforcement Learning

Jonas Keller

Supervised by

Prof. Dr. Ostap Okhrin

Prof. Dr. Pascal Kerschke

Dipl. Wi.-Ing. Martin Waltz

Dresden, 26. August 2024

Abstract

With the combination of Reinforcement Learning (RL) and Artificial Neural Networks (ANNs), Deep Reinforcement Learning (DRL) agents are shifted towards being non-interpretable black-box models. Developers of DRL agents, however, could benefit from enhanced interpretability of the agents' behavior, especially during the training process. Improved interpretability could enable developers to make informed adaptations, leading to better overall performance. The explainability methods Partial Dependence Plot (PDP), Accumulated Local Effects (ALE) and SHapley Additive exPlanations (SHAP) were considered to provide insights into how an agent's behavior evolves during training. Additionally, a decision tree as a surrogate model was considered to enhance the interpretability of a trained agent. In a case study, the methods were tested on a Deep Deterministic Policy Gradient (DDPG) agent that was trained in an Obstacle Avoidance (OA) scenario. PDP, ALE and SHAP were evaluated towards their ability to provide explanations as well as the feasibility of their application in terms of computational overhead. The decision tree was evaluated towards its ability to approximate the agent's policy as a post-hoc method. Results demonstrated that PDP, ALE and SHAP were able to provide valuable explanations during the training. Each method contributed additional information with their individual advantages. However, the decision tree failed to approximate the agent's actions effectively to be used as a surrogate model.

Contents

List of Figures v

List of Tables vii

List of Abbreviations ix

1 Introduction 1

2 Foundations 3

2.1 Machine Learning . . . 3

2.1.1 Deep Learning . . . 3

2.2 Reinforcement Learning . . . 4

2.2.1 Markov Decision Process . . . 5

2.2.2 Limitations of Optimal Solutions . . . 9

2.2.3 Deep Reinforcement Learning . . . 10

2.3 Explainability . . . 11

2.3.1 Obstacles for Explainability Methods . . . 11

3 Applied Explainability Methods 13

3.1 Real-Time Methods . . . 13

3.1.1 Partial Dependence Plot . . . 13

3.1.1.1 Incremental Partial Dependence Plots for Dynamic Modeling Scenarios . . . 15

3.1.1.2 PDP-based Feature Importance . . . 16

3.1.2 Accumulated Local Effects . . . 16

3.1.3 SHapley Additive exPlanations . . . 19

3.2 Post-Hoc Method: Global Surrogate Model . . . 22

4 Case Study: Obstacle Avoidance 23

4.1 Environment Representation . . . 23

4.2 Agent . . . 25

4.3 Application Settings . . . 25

5 Results 29

5.1 Problems of the Incremental Partial Dependence Plot . . . 29

5.2 Real-Time Methods 30
5.2.1 Feature Importance 30
5.2.2 Computational Overhead 33
5.3 Global Surrogate Model 34

6 Discussion **39**

7 Conclusion **43**

Bibliography **xi**

Appendix **xiii**

A Incremental Partial Dependence Results **xv**

List of Figures

2.1 Schematic representation of an artificial neural network. 4
2.2 Markov Decision Process. 6

3.1 incremental PDP 16
3.2 Calculation of Accumulated Local Effects 19

4.1 Schematic representation of "Simple-OA" environment. 24

5.1 Individual Partial Dependence Plots (PDPs) 31
5.2 Feature importance based on the Partial Dependence Plot (PDP) 32
5.3 Individual plots of Accumulated Local Effects (ALE) 33
5.4 Feature importance based on the Accumulated Local Effects (ALE) . . . 34
5.5 Individual SHapley Additive exPlanations (SHAP) Dependence Plots . . 34
5.6 SHapley Additive exPlanations (SHAP)-based feature importance 35
5.7 Runtimes of the explainability methods 35
5.8 Sampled environments for the decision tree's evaluation. 37
5.9 Comparison between agent's actions and decision tree's outputs in the first sampled environment. 38
5.10 Comparison between agent's actions and decision tree's outputs in the second sampled environment. 38

A.1 Individual iPDPs xvi
A.2 iPDP feature importance xvii

List of Tables

5.1 Runtime of iPDP 29
5.2 Training results of the decision tree 36

List of Abbreviations

AI	Artificial Intelligence
ALE	Accumulated Local Effects
ANN	Artificial Neural Network
DDPG	Deep Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
iPDP	incremental PDP
MAE	Mean Absolute Error
MDP	Markov Decision Process
ML	Machine Learning
MSE	Mean Squared Error
OA	Obstacle Avoidance
PDP	Partial Dependence Plot
RL	Reinforcement Learning
SHAP	SHapley Additive exPlanations

1 Introduction

The fast development of Artificial Intelligence (AI) in recent years led to a widespread application of algorithmic methods the daily life [1]. Especially ANNs has been applied extensively in both industry and research in a variety of areas. DRL, as a subfield of AI, is the combination of RL and ANN. This method focuses on learning autonomously from interaction with an environment and has been successfully applied in areas like autonomous driving [13], robotics [19] or finance [11].

The integration of ANNs into RL resulted in a remarkable performance increase [10]. However, applying more complex methods shifts DRL models towards being non-interpretable black-box models. In critical applications where it is essential to justify an agent's decisions, DRL would benefit from better explainability and therefore interpretability. Furthermore, DRL could benefit from a higher level of interpretability before it is applied in real-world scenarios. During the training process, an agent adapts its policy based on the interaction with the environment. The reasons for these adaptations, however, are mostly not comprehensible for humans. A developer of a DRL agent could benefit from gaining insights into the agent's policy during the training process. Being able to interpret the changes in an agent's decision making process already during the training could enable the developer to make adaptations leading to a better performance of the agent.

Therefore, this work aims to investigate the ability of explainability methods to provide insights into an agent's policy during the training process and after the agent has been trained.

In the following, foundations of RL, DRL and explainability methods will be described. After that, the chosen methods will be explained in detail and applied in a case study. The ability to provide explanations as well as the computational costs will be evaluated and discussed in the end.

2 Foundations

2.1 Machine Learning

"[Machine Learning (ML)] is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions." [4].

A program coded with traditional programming techniques would need a long list of rules to be able to solve problems with a complexity level. Every single one of these rules would need to be defined manually. Most often, this leads to large programming effort and confusing code. In contrast, ML algorithms can learn their own rules independently and can be coded in shorter and arguably simpler programs. [6]

There are many ways to classify different methods of ML. One common criteria for classification is the amount and type of supervision during the training process. Methods of Supervised and Unsupervised Learning are trained with a dataset. Supervised Learning methods need a training dataset consisting of labeled data. This means that the algorithms get the desired outcome for every data point in the training data set. The training data set for Unsupervised Learning methods is unlabeled. Therefore, the ML algorithms have to learn from the data on their own. RL methods learn from interaction with their environment instead of an already configured dataset. Latter methods will be described in section 2.2 (Reinforcement Learning). [6]

2.1.1 Deep Learning

Deep learning is a subset of ML. Its main subject is the study and use of ANNs as the learning component of the algorithms. [6]

ANNs were created by inspiration of biological neural networks [12]. ANNs consist of artificial neurons which are arranged as layers. One network consists of an input layer and an output layer. An arbitrary amount of additional layers called hidden layers can be arranged in between these two layers. Hidden layers can consist of an arbitrary amount of neurons. An ANN containing multiple hidden layers is called a deep neural network and the study of these networks is the field of deep learning. [6]

Fully connected layers are used in classical ANN. Each neuron of such a layer is connected to each neuron of the previous layer. Each created connection between two neurons is assigned a variable factor called weight.

All information are processed by an ANN as numeric values. When a single data point is passed into an ANN, its information is received through the input layer and transmitted to the first hidden layer. This layer processes the information and then passes it to the next layer through the weighted connections. This process gets repeated until the information reach the output layer. The output layer then outputs the ANN's prediction for the current data point. [6]

A schematic representation of an ANN is shown in Figure 2.1.

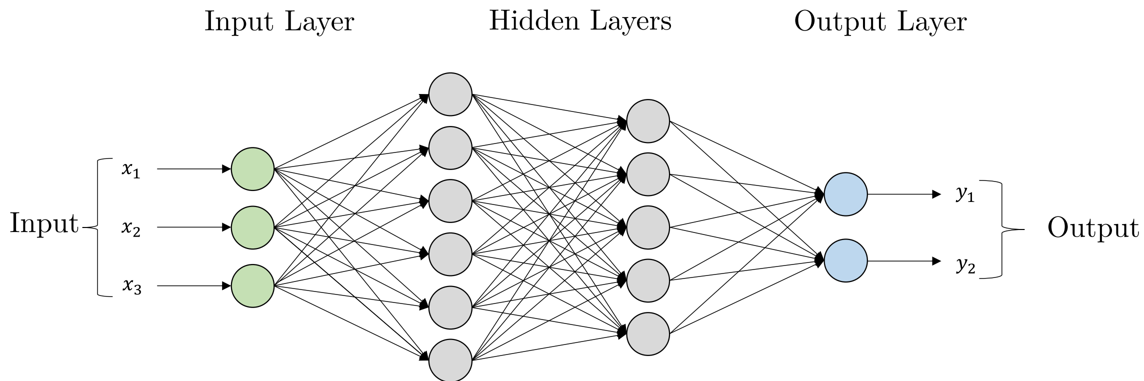

Figure 2.1: Schematic representation of an artificial neural network. It consists of an input layer, two hidden layers and an output layer. They are fully connected layers since every neuron is connected to every neuron of the previous layer. Information passes the network starting from input layer to output layer.

In case of supervised learning, an ANN gets trained with the backpropagation algorithm. A training step starts with a forward pass of a data point's information through the ANN from the input layer to the output layer. The output layer makes a prediction for the current data point. The prediction gets compared with the corresponding label of the data point and the difference of both gets computed. This difference represents the error which should be minimized. Subsequently, the ANN gets passed backwards. During this backwards pass the contribution to the error of every weighted connection gets computed. The weights of the connections then get adapted so that the overall error is minimized. These steps are repeated for every data point in the training data set. [6]

2.2 Reinforcement Learning

As mentioned in section 2.1 (Machine Learning), RL methods learn from interaction with the environment they are applied in. Algorithms of this kind are inspired by the natural learning of living beings. During its learning process, a living individual is aware of how its environment responds to its actions. It seeks to influence what happens through its behavior to achieve its goals. In analogy, a so called RL agent makes observations of its environment, takes actions and receives rewards in return. Its objective is to maximize the received reward over time. [24]

RL algorithms can be applied to a wide variety of tasks. For example, an agent can be applied to control a robot. In this case, the real world is the environment which is observed by the agent through sensors. The agent's actions are signals to activate the robot's motors. The agent receives rewards whenever the robot's task is successfully fulfilled. [6]

RL agents can also be applied in a variety of games, ranging from physical board or card games to all sorts of video games. In these cases, the environment is the game itself with its rules and boundaries. An agent can observe the environment through parameters describing the state of the game or screenshots in case of video games. The agent's actions are the possible moves a player can execute in the specific game. The agent receives rewards according to the game's objectives or ranking metrics. [6]

Other examples for the application of RL agents could be to observe stock market prices and decide when to buy or sell or to observe a room's temperature and decide whether to use heating or not. [6]

Every RL system comprises the same main components. As mentioned before in this section, an *agent* is the active decision-making element of a RL system. It has explicit goals which it seeks to achieve through its actions. The *environment* is the setting in which the agent takes its actions. The environment changes as a response to the agent's actions. A *policy* determines the agent's behavior at a given time and is therefore the core of an RL system. "[It] is a mapping from perceived states of the environment to actions to be taken when in those states." [24]. Policies can range from being a simple function or lookup table to involving complex computations. At any time step the environment sends to the agent a *reward*, which consists of a single number. The agent's objective is to maximize the total reward it receives over time. Therefore, the reward is a measurement for the quality of events. During an agent's learning phase, the policy is primarily changed on the basis of the reward. A *value function* assigns a value to each state the agent can be in. The value is a measurement for the quality of a state in the long run. "[It] is the total amount of reward an agent can expect to accumulate over the future, starting from that state." [24]. Optionally, a RL system can have a *model of the environment*. A model allows inference about how the environment behaves after executing an action in a certain state. RL methods that make use of a model are called model-based methods, methods not making use of a model are called model-free methods. [24]

2.2.1 Markov Decision Process

RL can be formalized in the context of Markov Decision Processes (MDPs). They are an idealized mathematical framework that can be used for modeling the RL problem.

In practice, RL seeks the "optimal control of incompletely-known [MDP]" [24].

MDPs formalize sequential decision making processes, where actions influence immediately received rewards, but also subsequent states and therefore possible future rewards. The decision maker and learner is the agent. It interacts continually with the environment which comprises everything outside of the agent. When the agent takes an action, the environment changes which then leads to a new situation. This gives rise to two elements which the agent receives as an answer from the environment. The agent receives a representation of the environment's state and a reward in form the of a numeric value. The agent then evaluates the received state and reward and chooses the next action. This sequential agent-environment interaction happens at discrete time steps. The agent's objective is to maximize the received reward over time. A schematic representation of the sequential agent-environment interaction in a MDP is shown in figure 2.2. [24]

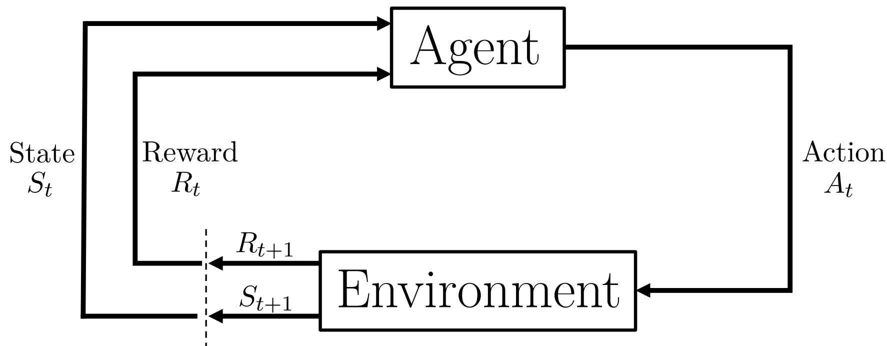

Figure 2.2: The continual interaction cycle between the agent and the environment in a Markov Decision Process. At a current time step t, the agent receives a representation of the environment's state S_t and a reward R_t. It evaluates these two parameters and takes an action A_t. This leads to a change of the environment, a representation of a new state S_{t+1} and a new reward R_{t+1} at the subsequent time step $t+1$. .The upper case letters represent random variables. Figure adapted from [24].

In a finite MDP, the sets of possible states $\mathcal{S}$, actions $\mathcal{A}$ and rewards $\mathcal{R}$ have a finite number of elements. At a time step t, the variables $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}$ and $R_t \in \mathcal{R}$ are treated as random variables. In case of a finite MDP, R_t and S_t have well defined discrete probability distributions. Realizations of these random variables depend only on the preceding state and action. "That is, for particular values of these random variables, $s' \in \mathcal{S}$ and $r \in \mathcal{R}$, there is a probability of those values occurring at time t, given particular values of the preceding state and action":

$$p\left(s', r \mid s, a\right) \coloneqq \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\}, \qquad \forall s', s \in \mathcal{S}, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}\left(s\right). \tag{2.2.1}$$

The function p defines the environment's dynamics in an MDP such that the probability of each possible value for state S_t and reward R_t depend only on the preceding state S_{t-1} and action A_{t-1}.

So far the agent's objective was described informally as the maximization of the received reward in the long run. For a formal definition of the objective the return G_t is introduced as "some specific function of the reward sequence"[24]. The agent's objective is to maximize the expected return. The simplest return function is the sum of the rewards:

$$G_t \coloneqq R_{t+1} + R_{t+2} + R_{t+3} + \dots \tag{2.2.2}$$

This return function makes sense if the agent-environment interaction breaks into subsequences. In that case, every sequence has a final time step and consists therefore of a finite number of total time steps. Those cases are called episodic tasks. If the agent-environment interaction goes on continually without a final time step, the return function from 2.2.2 could lead to an infinite return. These cases are called continuing tasks. The concept of discounting with a discount rate γ is introduced to prevent infinite returns. In this approach, an agent seeks to maximize the expected discounted reward:

$$G_t \coloneqq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad \gamma \in [0, 1]\,, \tag{2.2.3}$$

The formulation of the discounted return function holds for episodic and continuing tasks if some conventions are introduced. The description of these conventions is beyond the scope of this work and can be found in [24].

Returns at successive time steps are related to each other in a recursive manner:

$$\begin{aligned} G_t &\coloneqq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.2.4}$$

As mentioned earlier in this section, a value function assigns a value to each state the agent can encounter. It is a measurement for the quality of a state in the long run. The quality is expressed in expected return starting from a certain state. The expected return depends on the actions the agent will take in future time steps. Therefore, value functions are defined with respect to policies . As mentioned earlier in this section, a policy "is a mapping from perceived states of the environment to actions to be taken when in those states." [24]. A policy $\pi(a|s)$ defines the probability that at a time step t action a is selected if the agent is in state s. The objective of RL methods is to modify

the policy as a result of their experience. The value function $v_\pi(s)$ of a state s under a policy π "is the expected return when starting in s and following π thereafter" [24]:

$$v_\pi(s) \coloneqq \mathbb{E}_\pi\left[G_1 \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \qquad \forall s \in \mathcal{S} \tag{2.2.5}$$

Where $\mathbb{E}_\pi[\cdot]$ represents the expected value of a random variable given that the agent follows policy π.

A relationship between the value of a state and the value of its successor states is expressed by the Bellman equation [3]:

$$\begin{aligned} v_\pi(s) &\coloneqq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right], \qquad \forall s, s' \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s) \end{aligned} \tag{2.2.6}$$

This consistency condition holds for any policy π followed by an agent, any state s and the value of its successor states s'. The Bellman equation makes use of recursive relationships of the value functions similar to the return in formula 2.2.4. "The value function v_π is the unique solution to its Bellman equation" [24]. Solving the Bellman equation 2.2.6 to receive a representation of the value function is used in a number of RL methods.

As mentioned before in this section, the objective of RL is to modify the policy such that the received reward is maximized in the long run. Policies can be compared to each other with value functions. "A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states" [24]:

$$\pi \geq \pi' \;\Leftrightarrow\; v_\pi(s) \geq v_{\pi'}(s), \qquad \forall s \in \mathcal{S} \tag{2.2.7}$$

A policy that is better than or equal to all other policies is an optimal policy π_*. It is defined under the optimal state-value function v_*:

$$v_* \coloneqq \max_\pi v_\pi(s), \qquad \forall s \in \mathcal{S} \tag{2.2.8}$$

Formulating the Bellman equation 2.2.6 for the optimal state-value function 2.2.8 leads to the Bellman optimality equation:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi_*}\left[G_t \mid S_t = s, A_t = a\right] \\ &= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a\right] \\ &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*\left(S_{t+1}\right) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s', r} p\left(s', r \mid s, a\right)\left[r + \gamma v_*\left(s'\right)\right] \end{aligned} \tag{2.2.9}$$

It is written without reference to a specific policy . Since v_* is an optimal value function, the policy in 2.2.6 can be replaced by the best possible action.

The Bellman optimality equation is a system of equations with one equation for each state. In case of a finite MDP, the Bellman optimality equation system can be solved for a unique solution. An optimal policy can then be determined from the resulting optimal value function v_*. "Any policy that is greedy with respect to the optimal evaluation function v_* is an optimal policy" [24].

2.2.2 Limitations of Optimal Solutions

Solving the Bellman optimality equation leads to an optimal solution of an RL problem and is therefore an important mathematical concept. However, this method relies on assumptions which can rarely be met in reality. For example, the environment's dynamics have to be known completely for an MDP to be solvable completely. Additionally, the computations have to be solvable in an appropriate amount of time. However, already small state and action spaces lead to computational efforts that exceed these limits. Therefore, many methods that solve RL problems can be understood as approximately solving the Bellman optimality equation. These methods use experience from continual agent-environment interaction instead of complete knowledge of the environment's dynamics. [24]

To do so, almost all RL methods follow the iterative framework of generalized policy iteration. This consists of the two alternating processes of policy evaluation and policy improvement. At the start, a value function and a policy are assigned randomly. During a policy evaluation step, the value function is made consistent with the current policy. During a policy improvement step, the policy is made greedy with respect to the current value function. These two alternating processes iterate till they theoretically converge to an optimal policy and optimal value function or till the policy and value function exceed a defined threshold. [24]

In case of a finite MDP scenario with small enough state and function spaces, the value functions can be completely represented as tables. Every state or state action pair is one table entry. RL algorithms solving such problems are referred to as tabular

solution methods.

Many real world decision making scenarios, however, have very large state spaces. They are too large to be represented as tables or might even be infinite. For an RL method to be able to make appropriate decisions in scenarios with large state spaces, it has to be able to generalize from states it already had encountered. To do so, one out of many function approximation methods are applied. RL algorithms applied to these problems are referred to as approximate solution methods. [24]

2.2.3 Deep Reinforcement Learning

Approximate solution methods include a variety of different approaches for function approximations to generalize from already learned knowledge. Both linear and nonlinear function approximation can be used for this purpose. In particular, ANNs are often used as nonlinear function approximators. The field that studies the combination of RL methods with deep learning is referred to as DRL. [24]

Instead of representing the value function $v_\pi(s)$ in a tabular form, DRL methods represent the value function in a parameterized form $\hat{v}(s, \mathbf{w})$ [24]. The weight vector $\mathbf{w} \in \mathbb{R}^d$ consists of the connection weights of the ANN. These DRL algorithms are called value based methods [22]. Other methods approximate the policy function directly without computing the value function first. These methods are referred to as policy methods [22].

An advantage of DRL methods over classical RL methods is the ability of being applied in problems with large action and value spaces. The value function and policy are in parameterized form rather than stored in a table. Therefore, the necessary space in memory depends on the number of used parameters and not on the number of states. Additionally, DRL can be applied in problems with high-dimensional state representations. ANNs are able to discover hierarchical structures in data automatically. This enables DRLs methods to process raw sensory data as state representations. A state representation, for example, can be camera images. In this case, the dimensionality of the state representation is the number of pixels in the image. [22]

A disadvantage of DRL and the scenarios they are applied in is that the environments are nonstationary leading to nonstationary policies as well. The dynamics of the environment can therefore change over time. This leads to a change in the optimal policy which a DRL method seeks to approximate. Additionally, the computational costs to train DRL methods can be high especially in problems with high-dimensional state representations. A state that is represented by a high number of numerical values needs to be processed by an ANN with a large number of neurons. This leads to a large number of weights that need to be adjusted during the training process. Furthermore, the interpretation of DRLs methods is difficult resulting from the usage of ANNs. Since the number of neurons and therefore the number of trainable weights can become very

large, it is difficult to backtrack the reasons for certain decisions taken by a DRLs agent. [22]

2.3 Explainability

ML models like ANNs can become very complex already at small number of layers. They contain a large number of variables that adapt to a dataset during a training process. With an increasing number of adapting parameters, the model becomes more complex and therefore more difficult to interpret by a human. Explainability methods aim to provide explanations about black box models and help human users to interpret them. The term explainability methods include a variety of methods that aim to extract "relevant knowledge from a machine-learning model concerning relationships either contained in data or learned by the model" [20].

The different methods can be classified according to a variety of taxonomies. A possible distinction can be made between local and global explainability methods. Global methods provide explanations about the entire model behavior whereas local methods explain individual predictions. Another distinction can be made between model-specific and model-agnostic methods. Model-specific methods use internals of the ML model to provide explanations and are limited to specific model classes. Model-agnostic methods, on the other hand, can be applied on any trained ML model. They have no access to the model's internals. Furthermore, ML models can be intrinsically interpretable. A ML model is considered to be interpretable if its complexity is low enough for a human to comprehend the model's predictions. [18]

2.3.1 Obstacles for Explainability Methods

A variety of obstacles have to be considered when applying explainability methods to a black box model. The application of explainability methods can be computationally expensive. Most model-agnostic methods, for example, depend on repeatedly computing predictions from the black box model on a data set. Therefore, the number of computations can become large for a large data set. This obstacle is especially severe when an explainability method is applied during a training process since it increases the overall training time. Furthermore, explainability methods themselves can be complex. A complex computation of results makes the interpretation difficult for a human.

The application of explainability methods in RL scenarios introduces domain specific obstacles. As mentioned before, most model-agnostic methods rely on computing predictions on a dataset. In RL, however, a preconfigured dataset is not available. Additionally, various explainability methods need labeled data to compute their results. However, assigning labels to states, which are the data in a RL scenario, is not possible.

The aim of this work is to choose explainability methods appropriate for a RL scenario. Additionally, the individual settings of the methods should be chosen such that

the influence of the described obstacles are minimized.

3 Applied Explainability Methods

The explainability methods applied in this work are described in the following. These methods can be applied for regression and classification tasks. Furthermore, some of the methods have dedicated versions for numerical and categorical feature values. To simplify the descriptions, the methods will only be described in the context of regression tasks and numerical feature values.

3.1 Real-Time Methods

The described methods in this section will be applied during the training process of an RL agent. Therefore, they will be referred to as real-time methods.

3.1.1 Partial Dependence Plot

The PDP is a global model-agnostic method which was first introduced by Friedman et al. [5]. "The [PDP] shows the marginal effect one or two features have on the predicted outcome of a machine learning model" [18]. Therefore, this method aims to help understanding the relationship between a feature of interest and the outcome of an ML model. Since the PDP is a model-agnostic method, the ML model does not need to fulfill specific requirements. PDPs can be applied to any ML model that takes a data point of a dataset as input and produces an output based on that data point.

Let S be the set of features of interest and C be its complementary set, such that the union of the two sets reflects the whole set of features. The partial dependence function is defined as:

$$\hat{f}_S\left(x_S\right) \coloneqq \mathbb{E}_{X_C}\left[\hat{f}\left(x_S, X_C\right)\right] = \int \hat{f}\left(x_S, X_C\right) d\mathbb{P}\left(X_C\right) \tag{3.1.1}$$

where $\hat{f}$ is the prediction function of a ML model. x_S are the values of the features of interest and X_C the values of the features in C, here treated as random variables. $d\mathbb{P}\left(X_C\right)$ is the marginal probability density of the feature values in C.

The partial dependence function (equation 3.1.1) can be estimated by calculating averages over a dataset:

$$\hat{f}_S\left(x_S\right) = \frac{1}{n}\sum_{i=1}^{n} \hat{f}\left(x_S, x_C^{(i)}\right) \tag{3.1.2}$$

where $x_C^{(i)}$ are actual values for the features in C and n is the number of data points in

the dataset.

The following steps describe the procedure to calculate the estimated partial dependence function (equation 3.1.2) for a single feature of interest.

1. **Determine evaluation grid points.** For a feature of interest j, the maximum and minimum value in the dataset has to be found. These values represent the range of the evaluation grid points. Then, a grid resolution with a number of grid points has to be chosen. Finally, the values of the grid points are calculated equidistantly inside the range of the maximum and minimum feature value.

2. **Obtain synthetic data set.** For each data point in the original dataset, the value of the j-th feature is replaced by the values of the grid points. This results in a new data set with synthetic data points.

3. **Get model predictions.** Get the ML model's prediction for every data point in the newly created synthetic data set.

4. **Calculate the final PDP** The estimated partial dependence function (equation 3.1.2) can now be calculated with the above obtained model predictions on the synthetic dataset. The average over all model predictions have to be calculated for each grid point.

The resulting PDP for the j-th feature can be interpreted as the change of the average model prediction over the dataset when the j-th feature changes.

An advantage of the PDP is a clear and intuitive interpretation. It represents how a feature influences the ML model's prediction on average. The visualization of this influence for a single feature is simple and therefore quickly understandable for all kinds of audiences. [18]

A disadvantage of the PDP is that the features must be independent of each other. PDP can't distinguish between a real effect and an effect caused by a correlation with other features. If features are strongly correlated, the synthetic dataset on which the PDP is calculated might contain unlikely or even physically impossible combinations of feature values. Another disadvantage are the large computational costs. Calculating the PDP for a single feature involves obtaining the ML model's prediction for the whole dataset for each grid point. Furthermore, a feature's distribution should be taken into account for the PDP's interpretation. Regions with a small number of data points might be over interpreted compared to regions with more data points. Additionally, the PDP can not capture heterogeneous effects meaning that the feature has the same effect in positive and negative direction in the prediction for different data points. Since the PDP visualizes the average prediction, those effects could cancel each other out. [18]

3.1.1.1 Incremental Partial Dependence Plots for Dynamic Modeling Scenarios

Muschalik et al. [21] proposed an adaptation of the PDP for dynamic modeling scenarios. In these scenarios, an ML model is incrementally updated for every observed data point in a continuous data stream. The authors argued that recomputing the original PDP (equation 3.1.2) for every observed data point quickly becomes infeasible in terms of computational overhead. The proposed method, called incremental PDP (iPDP), "extends on the PDP to extract time-dependent feature effects in non-stationary learning environments" [21].

Instead of recomputing the PDP (equation 3.1.2), iPDP reutilizes previous evaluations of the PDP by computing an exponential moving average using the recursion:

$$\hat{f}_S^{PD}\left(x_{t,k}^S, t\right) \coloneqq (1-\alpha) \cdot \hat{f}_S^{PD}\left(x_{t-1,k}^S, t-1\right) + \alpha \cdot f_t\left(\tilde{x}_{t,k}^S, x_t^{\bar{S}}\right) \tag{3.1.3}$$

with $\hat{f}_S^{PD}\left(x_{t,k}^S, 0\right) \coloneqq 0$, the number of a grid point $k = 1, ..., m$ and smoothing parameter $0 < \alpha < 1$. $x_{t,k}^S$ refers to the k-th grid point at time t and $\tilde{x}_{t,k}^S$ refers to the k-th model evaluation point at time t.

The iPDP grid points $x_{t,k}^S$ are dynamically updated to address for a possible change in a feature's range. Similarly to the point-wise estimates (equation 3.1.3), the iPDP grid points are updated using an exponential moving average:

$$x_{t,k}^S \coloneqq (1-\alpha) \cdot x_{t-1,k}^S + \alpha \cdot \tilde{x}_{t,k}^S \tag{3.1.4}$$

at each time step t for $k = 1, ..., m$ and $x_{0,k}^S \coloneqq 0$.

As mentioned in section 3.1.1 (Partial Dependence Plot), it is common practice to evaluate the PDP on equidistant grid points inside a range $[minimum, maximum]$ derived from a feature's distribution. To address the possible change in a feature's range, the minimum $x_{t,min}^S$ and maximum $x_{t,max}^S$ model evaluation points based on a feature's distribution are maintained over a recent time frame. The equidistant model evaluation points are then given as:

$$\tilde{x}_{t,k}^S \coloneqq x_{t,min}^S + \frac{k-1}{m-1}\left(x_{t,max}^S - x_{t,min}^S\right) \tag{3.1.5}$$

at each time step t and $k = 1, ..., m$. The process of updating the previous iPDP with the current model evaluations is illustrated in figure 3.1

The interpretation as well as the advantages and disadvantages of the iPDP are inherited from the standard PDP. However, an iPDP specific advantage is that the partial dependence function is updated at every time step reusing previous evaluations. This eliminates the need of recomputing the PDP for every time the model has adapted to new data.

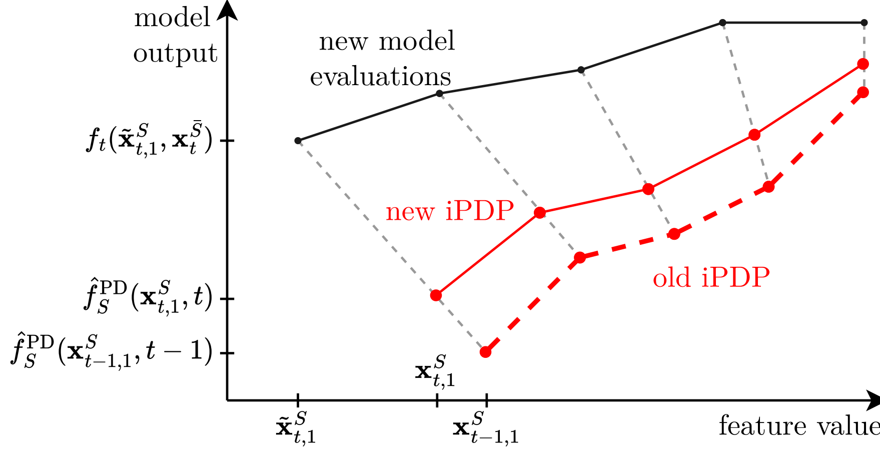

Figure 3.1: Illustration of the updating process of the iPDP. The new incremental PDP (iPDP) (red, solid) is updated by computing an exponential moving average of the old iPDP (red, dashed) and the new model evaluations (black). Figure adapted from [21].

3.1.1.2 PDP-based Feature Importance

Greenwell at al. [8] proposed a standardized, PDP-based feature importance measurement. Its motivation is that "any [feature] for which the PDP is "flat" is likely to be less important than those [features] whose PDP varies across a wider range of the response" [8].

The sample standard deviation is used to quantify the "flatness" of a PDP:

$$I\left(x_S\right) = \sqrt{\frac{1}{K-1}\sum_{k=1}^{K}\left(\hat{f}_S\left(x_S^{(k)}\right) - \frac{1}{K}\sum_{k=1}^{K}\hat{f}_S\left(x_S^{(k)}\right)\right)^2} \tag{3.1.6}$$

where $\hat{f}_S$ is the partial dependence function, K is the number of grid points the PDP is evaluated at and $x_S^{(k)}$ are the values of the grid points.

This measure was introduced to extract feature importance values from PDPs. ALE, which will be described in the following section (3.1.2 (Accumulated Local Effects)) visualize the effect of a feature on the model output in a closely related manner. A "flat" ALE plot indicates that the corresponding feature is less influential on the model's output than a feature whose ALE varies across a wider range. Therefore, the PDP-based feature importance measure is also applied to extract feature importances from ALE.

3.1.2 Accumulated Local Effects

The ALE is a global model-agnostic method which was introduced by Apley et al. [2] as an alternative to the PDP. "ALE describe how features influence the prediction of a ML model on average" [18]. ALE as a model-agnostic method has the same minimal

requirements for a ML model as PDP. Both methods can be applied to any model that takes a data point of a dataset as input and produces an output based on that data point.

Similar to the definition of the PDP (equation 3.1.1), let S be a set of features of interest and C be its complementary set, such that the union of the two sets reflects the whole set of features. The ALE function is defined as:

$$\begin{aligned}\hat{f}_{S,ALE}\left(x_S\right) &:= \int_{z_{0,S}}^{x_S} \mathbb{E}_{X_C|X_S=x_s}\left[\hat{f}^S\left(X_s, X_c\right)\middle|X_S = z_S\right] dz_S - \text{constant} \\ &= \int_{z_{0,S}}^{x_S} \left(\int_{x_C} \hat{f}^S(z_s, X_c) d\mathbb{P}(X_C|X_S = z_S)\right) dz_S - \text{constant}\end{aligned} \tag{3.1.7}$$

where $\hat{f}^S\left(x_s, x_c\right) = \frac{\partial \hat{f}(x_S, x_C)}{\partial x_S}$ denotes the partial derivative of the ML model's output function. Subtracting a constant centers the ALE plot so that the average effect is zero.

The ALE function (equation 3.1.7) can be estimated by calculating averages over a dataset. This results in the uncentered ALE function:

$$\hat{\tilde{f}}_{j,ALE}\left(x\right) = \sum_{k=1}^{k_j(x)} \frac{1}{n_j(k)} \sum_{i:x_j^{(i)} \in N_j(k)} \left[\hat{f}\left(z_{k,j}, x_{-j}^{(i)}\right) - \hat{f}\left(z_{k-1,j}, x_{-j}^{(i)}\right)\right] \tag{3.1.8}$$

with $\hat{\tilde{f}}_{j,ALE}\left(z_{0,j}\right) = 0$. j is the feature of interest, z are values of grid points, $k_j(x)$ is the number of grid points for feature j, $n_j(k)$ is the number of grid points in the k-th interval $N_j(k)$ and $\hat{f}\left(z_{k,j}, x_{-j}^{(i)}\right)$ is the prediction of the model when the feature j takes the value z_k and all other features are fixed at their values for the i-th data point.

The individual components of equation 3.1.8 can be broken down so that they reflect the name "Accumulated Local Effects". The subtraction of the model's prediction at adjacent grid points is the "Effect" the feature has for an individual instance in a certain interval. The inner sum adds up all effects within an interval which are then averaged by the number of data points in this interval. This accounts for the term "Local" in the name ALE. The left sum accumulates the effects across all intervals which represents the term "Accumulated". [18]

Estimating the ALE function (equation 3.1.8) from a dataset for the j-th feature consists of the following steps:

1. **Determine evaluation grid points.** The grid points at which the ML model will be evaluated can be determined by two major approaches. A grid resolution can be manually chosen. This leads to the same approach used to determine the grid points for the PDP calculation described in section 3.1.1 (Partial Dependence Plot).

   The grid points can also be determined by an algorithmic approach. An algorithm can be used to select the grid resolution and the values of grid points dependent

on the j-th feature's distribution in the data set. Values of grid points are selected such that the number of data points that fall into each interval in between two adjacent grid points does not fall below than a minimal threshold. This approach leads to a fine grained grid resolution while ensuring a minimum number of grid points in each interval and therefore a level of precision in the ALE estimation.

2. **Identify data points falling into the intervals.** For each interval between two adjacent grid points, the data points falling into this interval have to be identified. This process is illustrated in figure 3.2 with Feature 1 as feature of interest, grid points $z_{k,1}$ and Interval 4 as an example.

3. **Evaluate the differences of the model's predictions for each interval** For each data point that falls into a certain interval, the value of the j-th feature is replaced by the values of the data point's enclosing grid points (figure 3.2 solid horizontal lines between grid points $z_{3,1}$ and $z_{4,1}$). This results in pairs of data points in each interval. The ML model's predictions are then obtained for each pair and the difference of the two predictions is calculated.

4. **Accumulate the averaged differences.** For each interval, the prediction differences are averaged over the number of data points falling into an interval. The averaged prediction differences are then accumulated over all intervals. This finishes the process of calculating the estimated, uncentered ALE function (equation 3.1.8).

5. **Center ALE function** Finally, the ALE function is centered by subtracting it's average to center the mean effect:

$$\hat{f}_{j,ALE}\left(x\right) = \hat{\hat{f}}_{j,ALE}\left(x\right) - \frac{1}{n}\sum_{i=1}^{n}\hat{\hat{f}}_{j,ALE}\left(x_j^{(i)}\right) \tag{3.1.9}$$

   where n is the number of data points in the dataset and $\hat{\hat{f}}_{j,ALE}\left(x_j^{(i)}\right)$ is the value of the uncentered ALE function (equation 3.1.8) for the j-th feature value of the i-th data point.

The resulting ALE plot for the j-th feature can be interpreted as the "main effect of the feature at a certain value compared to the average prediction of the data" [18].

An advantage of ALE is that they are able to account for correlations between features. The method does so by calculating a feature's effect locally within an interval. PDP instead averages predictions over all possible values of the feature and therefore unlikely combinations of feature values. Furthermore, the computational costs of ALE are smaller compared to PDP. For n data points in a dataset, PDP needs to obtain n times the number of grid points predictions from the ML model. On the other hand, ALE needs to obtain the model's predictions only twice for each data point in the dataset. The reduced runtime enables a much finer evaluation grid and therefore more precise information about a feature's effect.

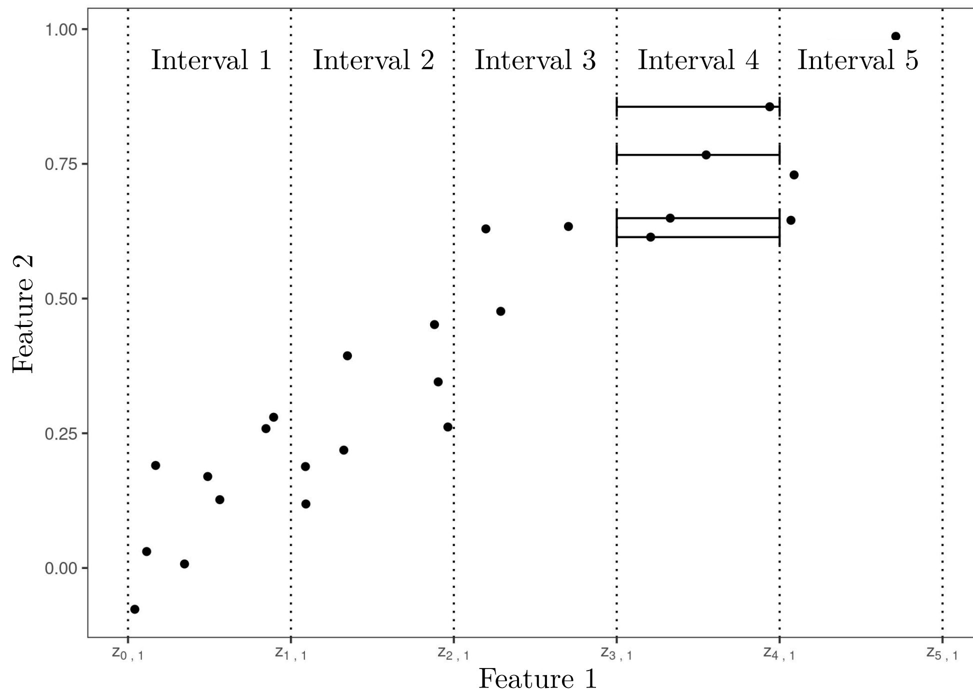

Figure 3.2: Illustration of the calculation of the Accumulated Local Effects (ALE) for an imaginary data set and Feature 1 as feature of interest. The data points are plotted by their values of Feature 1 and Feature 2. First step in the calculation is the division of Feature 1's range into intervals. Then, for each data point that falls into a certain interval (here Interval 4), the difference in the model's prediction is calculated by replacing the value of Feature 1 with the values of its enclosing grid points (here $z_{3,1}$ and $z_{4,1}$). The differences are then averaged inside each interval. Finally, the averaged differences are accumulated across all intervals and centered to compute the ALE curve. Figure adapted from [18].

3.1.3 SHapley Additive exPlanations

Other than the previously introduced methods, SHAP is a local model-agnostic method introduced by Lundberg et al. [16]. SHAP aims to explain a single prediction of an ML model by assigning each feature of the model's input an importance value for a particular prediction. To achieve that, SHAP approximates Shapley values which have their origin in coalitional game theory [23]. Each feature value of a data point can be seen as a "player" in a game. The coalition of all players achieves a payout which is the model's prediction for this data point. Shapley values then aim to "fairly distribute the "payout" among the features" [18]. The individual contributions to the prediction are calculated by forming coalitions of features and isolating the contributions.

The SHAP method represents the Shapley value explanations as a linear model:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j \tag{3.1.10}$$

where g is the explanation model, $z' \in \{0,1\}^M$ is the coalition vector, M is the number of features in the dataset and $\phi_j \in \mathbb{R}$ are the Shapley values, the contribution of feature j. The Shapley values need to be calculated with respect to a chosen baseline. Therefore, ϕ_0 is chosen to be the average prediction of the to be explained ML model over the dataset.

The coalition vector simulates the presence and absence of certain features. A positional entry of 1 represents that the corresponding feature is present and an entry of 0 represents that the corresponding feature is absent Therefore, for a data point x with all features present, the coalition vector x' consists only of 1's and formula 3.1.10 simplifies to:

$$g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j \tag{3.1.11}$$

KernelSHAP is an approximation approach for Shapley values proposed by Lundberg et al. [16]. It consists of the following steps to explain a single data point x [18]:

1. **Sample coalitions $z'_k \in \{0,1\}^M$, $k \in \{1, ..., K\}$.** In theory, K is the number of possible different coalitions that can be formed from the set of features. For ML models with a large feature set, however, a smaller number K can be chosen for the approximation process of SHAP. The coalitions are sampled with a probability distribution given by the SHAP kernel π_x (further described later in this section). The resulting set of sampled coalitions later becomes the training dataset for the linear explanation model g.

2. **Convert z'_k to original feature space with the mapping function $h_x(z'_k)$.** The mapping function $h_x(z'_k)$ is needed to convert the binary coalition vectors to data points with valid feature values. If a feature is assigned a 1, meaning it is present in the coalition, $h_x(z'_k)$ maps the 1 to the corresponding feature value of the data point x. Common ML models can not process a variable number of features as input. Therefore, the absence of a feature needs to be simulated. To achieve that, an absent feature, a feature that is assigned a 0, the mapping function maps the 0 to the feature value of another data point randomly sampled from the original data set.

3. **Get the prediction for each z'_k from the ML model $\hat{f}(h_x(z'_k))$.**

4. **Compute the weight for each z'_k with the SHAP kernel $\pi_x(z')$.** The SHAP kernel π_x assigns a weight to a coalition of features depending on the number of present features in the coalition $|z'|$ and the total number of features in the data set M:

$$\pi_x(z') := \frac{(M-1)}{\binom{M}{|z'|}|z'|\,(M-|z'|)} \tag{3.1.12}$$

The kernel assigns the largest weights to small and large coalitions. This follows the intuition that most of the information about a feature's effect can be obtained

by isolating the feature. A feature is most isolated in coalitions consisting of that feature only or consisting of all other features but it.

5. **Fit weighted linear explanation model g .** The linear explanation model g can then be calculated by optimizing the loss function:

$$L(\hat{f}, g, \pi_x) = \sum_{z' \in Z} \left[\hat{f}\left(h_x\left(z'\right)\right) - g\left(z'\right) \right]^2 \pi_x\left(z'\right) \tag{3.1.13}$$

The resulting coefficients ϕ_j from equation 3.1.13 are then the Shapley values of the j features for a data point x.

The method described above results in a Shapley value for each feature value of a single data point. The Shapley value ϕ_j for the j-th feature value can then be interpreted as the contribution of the feature value to the ML model's prediction in an additional manner. "The value of the j-th feature contributed ϕ_j to the prediction of this particular [data point] compared to the average prediction for the dataset" [18]. This additional interpretation is possible because the explanation model g in equation 3.1.11 is a linear regression model.

As mentioned before in this section, SHAP is a local model agnostic method aiming to explain a ML model's prediction for a single data point. In a subsequent publication, Lundberg et al. [15] proposed a global SHAP based feature importance measure derived from local SHAP values. The SHAP based feature importance value for the j-th feature is defined as the average absolute Shapley value across the whole data set:

$$I_j := \frac{1}{n} \sum_{i=1}^{n} \left| \phi_j^{(i)} \right| \tag{3.1.14}$$

where n is the number of data points in the data set and $\left| \phi_j^{(i)} \right|$ is the absolute Shapley value for the j-th feature of the i-th data point.

In the same publication, Lundberg et al. [15] introduced the SHAP dependence plot. The plot shows "how a feature's value [...] impacts the prediction [...] of every sample [...] in a dataset" [15]. Therefore, the SHAP dependence plot is a scatter plot which shows the feature values on the x-axis, the corresponding Shapley values on the y-axis and each point in the plot represents a data point in the data set. The authors argue that their plot provides richer information than the PDP (section 3.1.1 Partial Dependence Plot). While PDPs only show average effects, SHAP dependence plots additionally show the variance of Shapley values on the y-axis.

An advantage of the SHAP method is its clear interpretation. Furthermore, SHAP is a local-model agnostic method which can be extended to global explanations. While providing an overview about global feature importance, SHAP additionally delivers

more detailed explanations about individual predictions. [18]

A disadvantage of the above described KernelSHAP method is its high computational costs. The runtime of computing SHAP values increases drastically the bigger the dataset is and the more individual data points are explained. Furthermore, SHAP can not account for correlations between features. For correlated features, unlikely or physically impossible combinations of feature values might be used for the calculation. [18]

3.2 Post-Hoc Method: Global Surrogate Model

A global surrogate model is a model that is trained to predict the outputs of a black box ML model. The surrogate model is a method with a low complexity, which makes it an intrinsically interpretable model. Therefore, "the purpose of (interpretable) surrogate models is to approximate the predictions of the underlying [black box] model as accurately as possible and to be interpretable at the same time" [18]. Any interpretable model can be used as a surrogate model.

A training dataset is needed to train a surrogate model. Therefore, the predictions of the trained black box model on a chosen dataset are collected. These predictions then are the labels for the training dataset of the surrogate model.

The aim of a surrogate model is to approximate the predictions of the black box model as closely as possible. If the accuracy of the approximation is at a high level, the surrogate model can be interpreted and explanations can be concluded about the black box model's predictions.

A decision tree for regression is considered to be an interpretable model. Trained on a dataset, a decision tree splits the data multiple times such that data points are grouped together according to their labels. Each split depends on a single feature of the data and divides the data into two subsets. Every data point in the dataset belongs to one of the subsets resulting from the splitting.

The final subsets are called leaf nodes. The output of a decision tree is then the average target value of the training data falling into a certain leaf node. The number of leaf nodes depends primarily on the maximal depth of the tree.

With a rising number of leaf nodes, the decision tree becomes less interpretable. Therefore, the maximal depth has be chosen from a limited range to enable a decision tree to be applied as a surrogate model

4 Case Study: Obstacle Avoidance

The methods described in chapter 3 (Applied Explainability Methods) were evaluated in an exemplary case study. The goal of the application was to investigate the different methods' suitability to increase the understanding about an RL agent's policy. Additionally, the increase in computational costs that each method added were examined.

The case study was conducted on a RL setting introduced by Hart et al. [9]. In their publication, the authors investigated the influence of removing information about velocities on a DRL agent's performance. Therefore, multiple OA scenarios with different levels of complexity were introduced. In these scenarios, DRL agents were trained with the aim to avoid collisions with moving obstacles.

The "Simple-OA" environment was introduced for an "isolated analysis of the anticipation of a single trajectory" [9]. Therefore, the environment contains only two moving obstacles. Both obstacles move only in lateral direction, meaning their longitudinal speed is zero. Additionally, the lateral distance between the obstacles is constant over time. Therefore, both obstacles move with the same lateral speed. Both obstacles are initialized with the same longitudinal coordinate. The agent's goal is to pass in between the obstacles which was interpreted as a "moving finish line" [9]. A schematic representation of the "Simple-OA" environment is visualized in figure 4.1.

Furthermore, a traffic independent representation of the environment was chosen. This led to additional assumptions about the environment. The obstacles and the agent are represented as point masses and the agent's longitudinal speed is constant.

4.1 Environment Representation

The general representation of the environments described in the section above (chapter 4 Case Study: Obstacle Avoidance) consists of six features containing information about accelerations, speeds and positions about the agent and the obstacles. Considering a set of obstacles $\mathcal{M} = \{1, ..., N_{obstacles}\}$, with $N_{obstacles}$ as the total number of obstacles

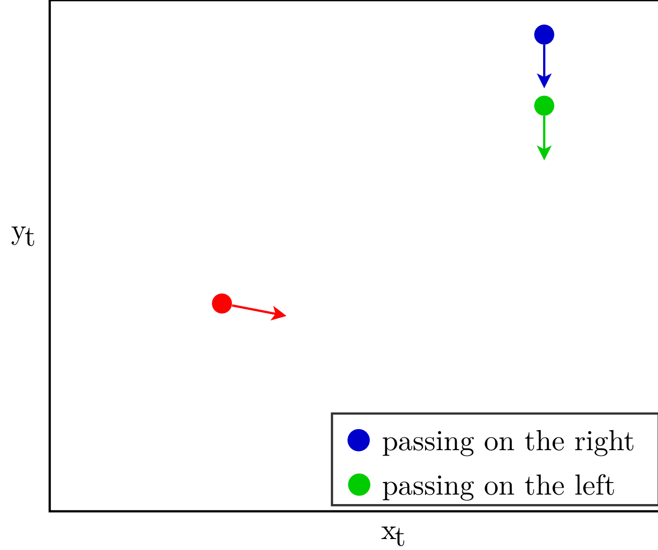

Figure 4.1: Schematic representation of the "Simple-OA" environment. The environment contains two moving obstacles (blue and green dots). Both obstacles move only in lateral direction (y-direction) with a constant speed. The relative distance between them is constant over time. Additionally, both obstacles are initialized with the same longitudinal coordinate (x-coordinate). The aim of the agent is to pass in between the obstacles, meaning it has to pass the blue obstacle on the right side of the agent's traveling direction and the green obstacles on the left side. This scenario can be interpreted as a "moving finish line". The figure is taken from [9].

in the environment, the state at timestep t is defined as:

$$s_t := \begin{pmatrix} \frac{\ddot{y}_{t,agent}}{a_{y,max}} \\ \frac{\dot{y}_{t,agent}}{v_{y,max}} \\ \frac{\dot{x}_{t,agent} - \dot{x_{t,i}}}{v_{x,max}} \\ \frac{\dot{y}_{t,agent} - \dot{y_{t,i}}}{v_{y,max}} \\ \frac{x_{t,agent} - x_{t,i}}{x_{scale}} \\ \frac{y_{t,agent} - y_{t,i}}{y_{scale}} \end{pmatrix} \tag{4.1.1}$$

where, at each timestep t, $x_{t,agent}$ and $x_{t,i}$ are the longitudinal position of agent and obstacle $i \in \mathcal{M}$. Respectively, $y_{t,agent}$ and $y_{t,i}$ are the lateral positions. $\dot{x}_{t,agent}$ and $\dot{x_{t,i}}$, $\dot{y}_{t,agent}$ and $\dot{y_{t,i}}$ denote the speed of the agent and obstacles in the corresponding directions and $\ddot{y}_{t,agent}$ denotes the agent's acceleration in lateral direction. $a_{y,max}$, $v_{x,max}$ and $v_{y,max}$ define the maximal value of the corresponding parameter and x_{scale} and y_{scale} are scaling parameters of the environment.

As mentioned in the section above (chapter 4 Case Study: Object Avoidance), the "Simple-OA" environment contains two obstacles, leading to $N_{obstacle} = 2$. Furthermore, the introduced constraints prevent relative movement between the two obstacles. This enables the reduction of the state representation s_t 4.1.1 to the number six features.

To simplify the understanding of the state representation, the individual features in 4.1.1 are rewritten:

$$s_t = \begin{pmatrix} \text{y} - \text{acceleration agent} \\ \text{y} - \text{speed agend} \\ \text{x} - \text{velocity difference agent} - \text{obstacle} \\ \text{y} - \text{velocity difference agent} - \text{obstacle} \\ \text{x} - \text{distance agent} - \text{obstacle} \\ \text{y} - \text{distance agent} - \text{obstacle} \end{pmatrix} \tag{4.1.2}$$

4.2 Agent

A DRL agent, that is trained in the above described scenario, can take an action consisting of a single value. The computed action $a_t \in [-1, 1]$ is then mapped to the agent's acceleration in lateral direction:

$$\ddot{y}_{t+1,agent} = \ddot{y}_{t,agent} + \Delta a_{y,max} \cdot a_t \tag{4.2.1}$$

where $\Delta a_{y,max}$ defines the maximal increase of the acceleration per time step.

A DDPG agent was chosen to be trained in the case study [14]. This algorithm is an adaptation of the Deep Q-Learning [17] to continuous action spaces. The agent was trained for 4000000 timesteps with a buffer size of 100000, an epoch length of 5000 and a batch size of 32. The training was repeated three times with the same parameters but different seed values to obtain more reliable results.

4.3 Application Settings

The explainability methods described in section 3.1 (Real-Time Methods) were applied during the agent's training process. Therefore, each method was applied with a frequency of 100000 time steps. The corresponding function for each method was computed using the states the agent has encountered in the previous 100000 time steps. This frequency was considered to be low enough compared to the overall time steps to add fine grained explanations. Furthermore, a set of 100000 states was considered to yield reliable results computed with the explainability methods. The computations were conducted on a high performance computing system and 10 CPUs were used.

The PDP, described in section 3.1.1 (Partial Dependence Plot), was calculated using the entire 1000000 previously encountered states at each evaluation. A grid resolution of five grid points was chosen to limit the additional computational costs. A PDP was plotted for each feature in the state representation.

The ALE, described in section 3.1.2 (Accumulated Local Effects), was also calculated using the entire 100000 previously encountered states at each evaluation. A variable

gird resolution calculated with an algorithm was chosen. The algorithm ensures that ten data points fall into each interval between two adjacent grid points. A much finer grid resolution compared to the PDP calculation was achieved which led to more detailed explanations. The additional runtime caused by the finer grid was comparable to the overhead introduced by the PDP calculation because the computational costs of the ALE calculation are lower than the costs of the PDP calculation. The ALE function was plotted for each feature in the state representation.

The SHAP method, described in section 3.1.3 (SHapley Additive exPlanations), was calculated using only 200 previously encountered states at each evaluation. A much smaller data set was chosen because SHAP introduces a severely bigger computational overhead than the PDP and the ALE. The 200 states were randomly sampled from the 100000 previously encountered states. A SHAP dependence plot was plotted for each feature in the state representation. Additionally, the SHAP-based feature importance was calculated.

The iPDP, described in section 3.1.1.1 (Incremental Partial Dependence Plots for Dynamic Modeling Scenarios), was updated at each time step following its incremental implementation. A grid resolution of five grid points was chosen following the reasons for the standard PDP. The time sensitivity value α from equation 3.1.3 was set to 0.001 The iPDP was plotted for each feature in the state representation with a frequency of 100000 time steps.

Additionally, based on the PDP and ALE computations, the feature importance measure described in section 3.1.1.2 (PDP-based Feature Importance) was calculated for both methods. The feature importance values were then scaled to the range [0, 1]. Furthermore, the values for feature importance were exponentially smoothed with $\alpha = 0.5$:

$$y_0 = x_0 \tag{4.3.1}$$

$$y_t = (1 - \alpha) y_{t-1} + \alpha x_t \tag{4.3.2}$$

where y is the smoothed value and x is the actual feature importance value.

The aim of the case study was to investigate how the different methods can add explanations about how the agent's decision making process changes over the progressing training time. Different visualizations across the explainability methods were obtained and their contribution to an overall understanding of the RL agent's policy were discussed. Furthermore, the computational costs in the form of an increase of runtime were examined. The runtime of PDP, ALE and SHAP was measured at each evaluation step. The computational overhead of the iPDP was compared to a training without applied explainability methods. Since the iPDP was updated at every time step, a dedicated time measurement was expected to lead to a significant increase of runtime itself.

A decision tree, described in section 3.2 (Post-Hoc Method: Global Surrogate Model), was applied after the RL was trained. Therefore, the last 100000 states the agent encountered during the training were used as a training dataset. The trained agent's actions on these states were used as the labels for the training data. The dataset was split into 80 % training data and 20 % test data. The decision tree was trained with maximal depths ranging from three to six. This depth range was considered to be appropriate for a decision tree to be interpretable by a human. Mean Squared Error (MSE) was chosen as a loss function.

The metrics MSE and Mean Absolute Error (MAE) were evaluated on the test data split for each resulting tree. Furthermore, the ability to adapt to the agent's decisions was evaluated on newly sampled environment configurations. Therefore, the trained agent was applied in these environment configurations. The agent's actions and the decision tree's outputs were then collected for every encountered state and compared.

The aim of applying a decision tree was to investigate it's ability to provide further explanations about a trained agent's decision making. Therefore, the decision tree's ability to adapt to the agent's policy was investigated.

5 Results

5.1 Problems of the Incremental Partial Dependence Plot

As mentioned in section 4.3 (Application Settings), the increase of runtime of the iPDP was compared to a training without the application of any explainability method. Three trainings were executed with and without the application of the iPDP but otherwise the same parameters. The seed value was varied between the three executions. The resulting computational overheads are listed in table 5.1.

Training Number	**With iPDP [h]**	**Without iPDP [h]**	**Difference [h]**
1	14.21	3.66	10.55
2	14.85	3.43	11.42
3	12.57	3.57	9.00
Average	13,88	3.55	10.32

Table 5.1: Runtime comparison between trainings with the application of the incremental Partial Dependence Plot (iPDP) and without. Both cases were executed three times with the same settings and varying seed values. The agents were trained for 4000000 time steps. The last line lists the average value of the corresponding column.

The iPDP led to an average increase of runtime of 10.32 h. Therefore, executing a training with the application of the iPDP took almost four times as long as the same training without any explanation methods.

The continuous application of an explainability method during the training of an RL agent always results in an increase of runtime. However, an explainability method's feasibility of application suffers from large computational overheads. Resulting from that, the iPDP's computational overhead was decided to be too large. Therefore, no further investigations about this method were conducted. Exemplary iPDP curves for the individual features at time step 2000000 (figure A.1) and feature importance results (figure A.2) for training number one are shown in the Appendix A

5.2 Real-Time Methods

5.2.1 Feature Importance

As described in section 4.2 (Agent), a DRL agent was trained three times with varying seed values. The explainability methods described in section 3.1 (Real-Time Methods) were applied during the training processes with a frequency of 100000 time steps. The results of one training are exemplary presented in the following. The individual results of each method are exemplary shown for time step 2000000 in figure 5.1, 5.3 and 5.5. The development of the feature importance over time based on each method is shown in figure 5.2, 5.4 and 5.6.

The individual PDPs for each feature in the state representation at time step 2000000 are shown in figure 5.1. The figure shows the partial dependence functions evaluated at five grid points. The horizontal lines at the bottom of each plot are the deciles. They divide the data set into ten equal parts giving a visualization of the data distribution.

The PDP of feature "y-distance agent-obstacle" (bottom left) and "y-speed difference agent-obstacle" (bottom center) show the largest influence on the agent's action. The remaining four features influence the agent's action within a similar lower range.

The value distribution of feature "y-speed agent" (top center), "x-distance agent-obstacle" (top left) and "y-distance agent-obstacle" (bottom left) is centered around certain points. This makes the interpretation less reliable in regions with less data points.

The feature importance based on the individual PDPs and calculated with equation 3.1.6 is shown in figure 5.2. According to the PDP calculations, the feature importance was calculated with a frequency of 100000 time steps over the total training time of 4000000 time steps. According to this measurement, the features "y-distance agent-obstacle" (gray, dashed) and "y-speed difference agent-obstacle" (green, dashed) were most important for the agent's decisions relative to the other features. The feature "y-distance agent-obstacle" had a higher importance than the other features from the first evaluation, staying at the same level over the whole training time. Feature "y-speed difference agent-obstacle" was given a low importance at the beginning of the training but reached and kept a higher level after 500000 time steps. The four remaining features had a low level of influence on the agent's action over the whole training time.

The individual ALE plots for each feature in the state representation at time step 2000000 are shown in figure 5.3. The figure shows the ALEs evaluated at a very high grid resolution. The resolution was calculated such that ten data points fall into each interval between two adjacent grid points. Most of the individual grid points are not distinguishable due to the high grid resolution and the limited space for visualization.

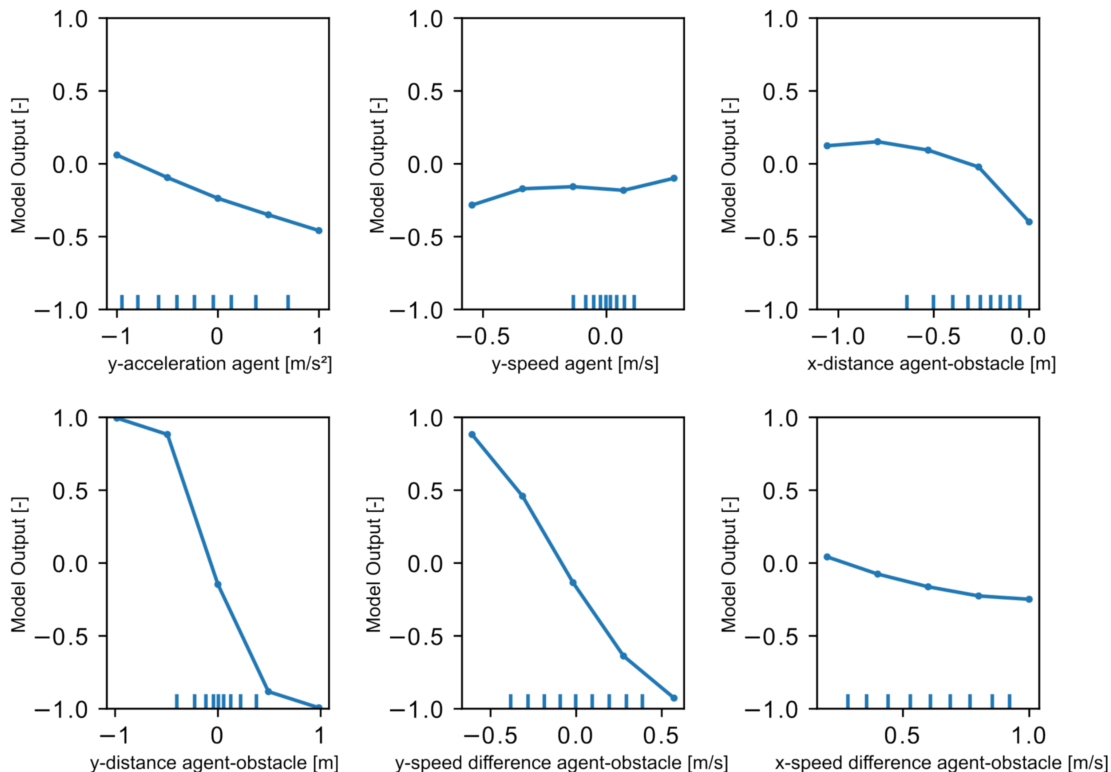

Figure 5.1: Individual Partial Dependence Plots (PDPs) for each feature in the state representation. The PDPs were calculated on 100000 data points. The partial dependence function was evaluated at five grid points. The horizontal lines at the bottom of each plot represent the deciles, meaning the lines divide the data set into ten equal parts.

The horizontal lines at the bottom of each plot represent the same deciles as in the PDPs in figure 5.1 since both methods were calculated with the same data set at every evaluation step

The ALEs show similar trends for the individual features compared to the PDPs. The effects of feature "y-distance agent-obstacle" (bottom left) and "y-speed difference agent-obstacle" (bottom center) are larger than the effects of the remaining features.

The feature importance based on the individual ALEs and calculated with equation 3.1.6 is shown in figure 5.4. The development of the importances over time are similar to the ones calculated on the PDPs (figure 5.2). The features "y-distance agent-obstacle" (gray, dashed) and "y-speed difference agent-obstacle" (green, dashed) are given an even higher importance relative to the other features compared to the importance based on the PDP. The remaining four features had a low level of influence on the agent's action over the whole training time.

The individual SHAP dependence plots for each feature in the state representation at time step 2000000 are shown in figure 5.5. 200 data points randomly sampled from

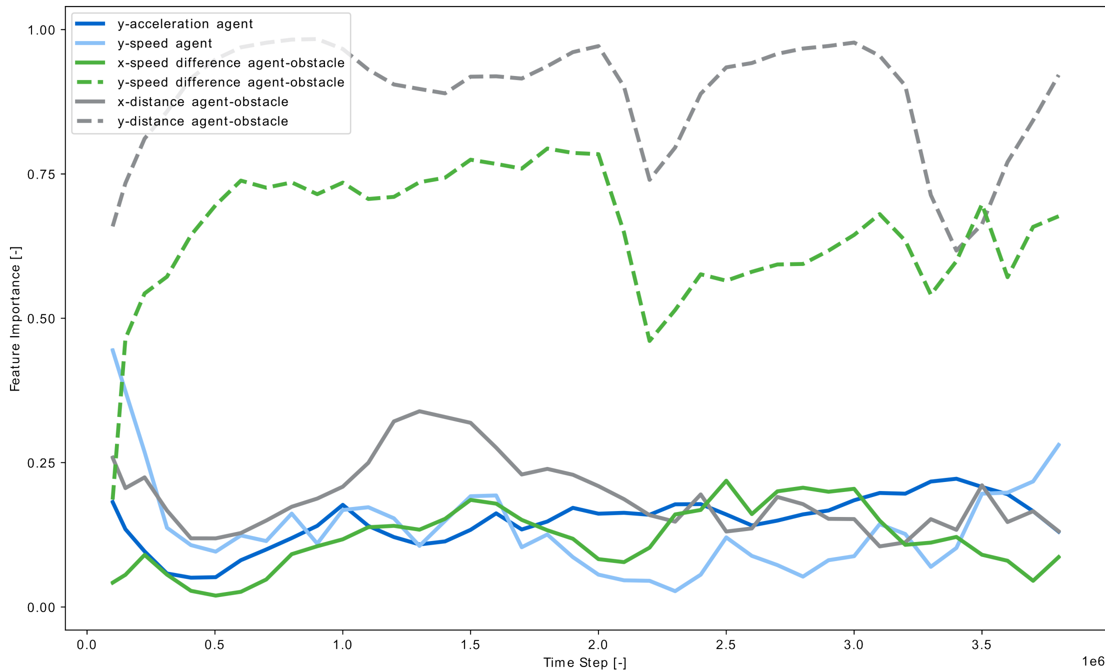

Figure 5.2: Partial Dependence Plot (PDP)-based feature importance during the training of an agent for 4000000 time steps. The PDPs were calculated with a frequency of 100000 time steps.

the 100000 previously encountered states were used to evaluate SHAP values. Each data point and its SHAP value corresponding to the different features are represented by a dot. Therefore, the data distribution is visualized directly by the x-location of the dots. Note that the range of the y-axis varies between the plots.

The dots of feature "y-distance agent-obstacle" (bottom left) and "y-speed difference agent-obstacle" (bottom center) show a clear trend from high SHAP values falling to low values when the feature value increases. This trend is similar to the trends in the plots for PDP (figure 5.1) and ALE (figure 5.3). Feature "y-acceleration agent" (top left) shows a similar trend but in a smaller range of total SHAP values. The plots of the remaining features do not show clear trends. Therefore, the corresponding feature values had no clear influence on their SHAP values.

The feature importance based on the individual SHAPs values and calculated with equation 3.1.14 is shown in figure 5.6 The development of the importances over time are similar to the ones calculated on the PDPs (figure 5.2) and the ALE 5.4. The features "y-distance agent-obstacle" (gray, dashed) and "y-speed difference agent-obstacle" (green, dashed) are still the most important ones relative to the other features over the whole training time. However, the relative importance compared to the other features is lower.

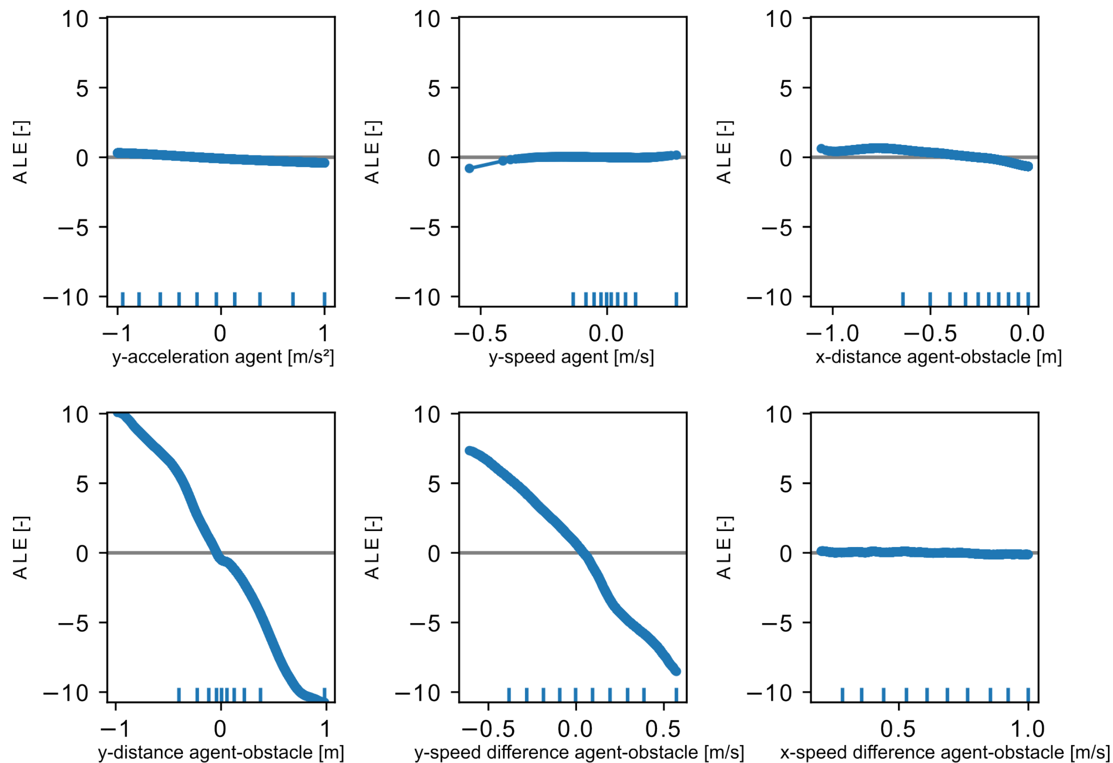

Figure 5.3: Individual Accumulated Local Effects (ALE) for each feature in the state representation. The ALEs were calculated on 100000 data points. The grid resolution was calculated such that ten data points fall in each interval between two adjacent grid points. Most grid points are not distinguishable because of a high grid resolution and small visualization space. The horizontal lines at the bottom of each plot represent the deciles, meaning the lines divide the data set into ten equal parts.

5.2.2 Computational Overhead

The runtime of each explainability method was measured at each evaluation step across the three different trainings. They are visualized as box plots in figure 5.7.

The method PDP had the lowest runtime on average. The average values are in an approximate range from 120 seconds to 140 seconds. The method ALE had the biggest runtime on average. The average values range approximately from 155 seconds to 180 seconds. As mentioned in section 3.1.2 (Accumulated Local Effects), ALE is less computational expensive than the PDP in general. The higher runtime here results from the large difference in grid resolution between the two methods. The average runtime values of the method SHAP range approximately from 150 seconds to 165 seconds. Therefore, they lay in between the average runtimes of PDP and ALE.

The variance of the runtimes of each method was influenced by the different training runs. In "Training 1", all methods had the lowest runtime variance while the variance was the highest in "Training 2".

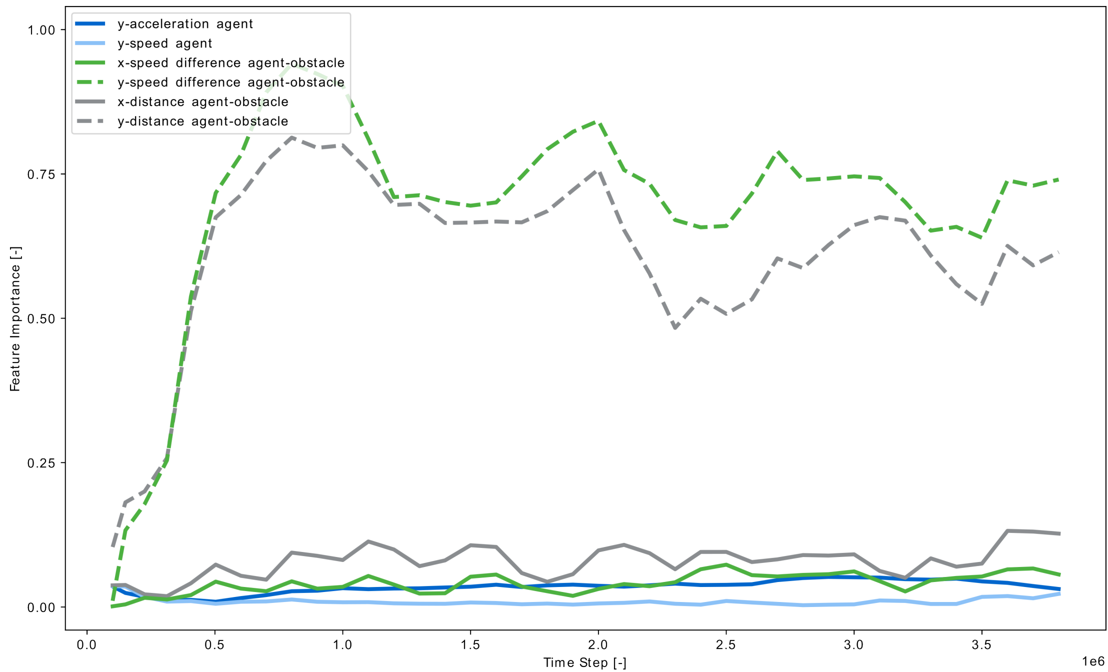

Figure 5.4: Feature importance calculated from the Accumulated Local Effects during the training of an agent for 4000000 time steps. The values were calculated with the formula from the Partial Dependence Plot (PDP)- based feature importance. The ALEs were calculated with a frequency of 100000 time steps.

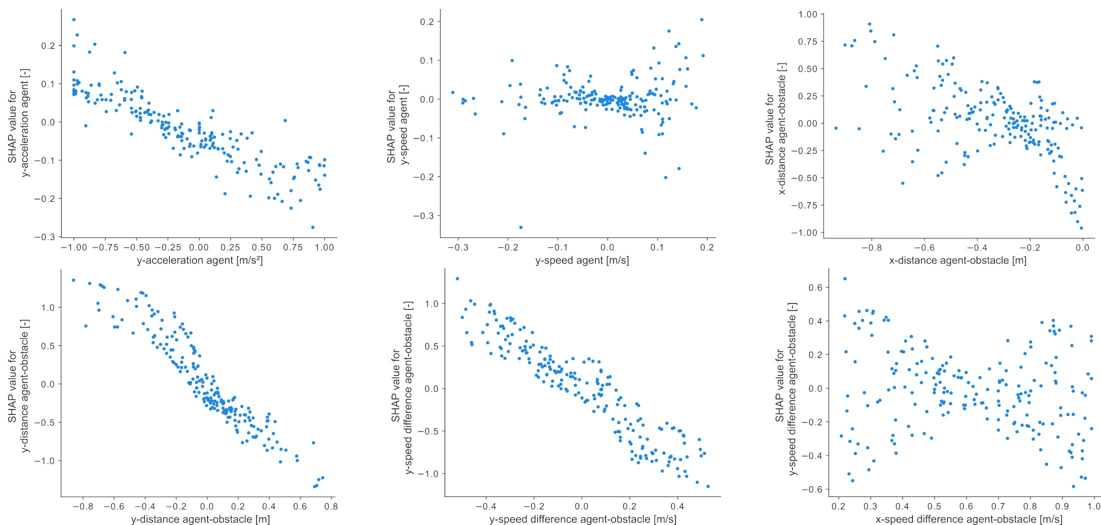

Figure 5.5: Individual SHapley Additive exPlanations (SHAP) Dependence Plots for the six features in the state representation. The SHAP values were calculated for 200 data points randomly sampled from 1000000 data points.

5.3 Global Surrogate Model

As described in section 4.3 (Application Settings), a decision tree was trained with maximal depths ranging from three to six. The results of the different trees for the

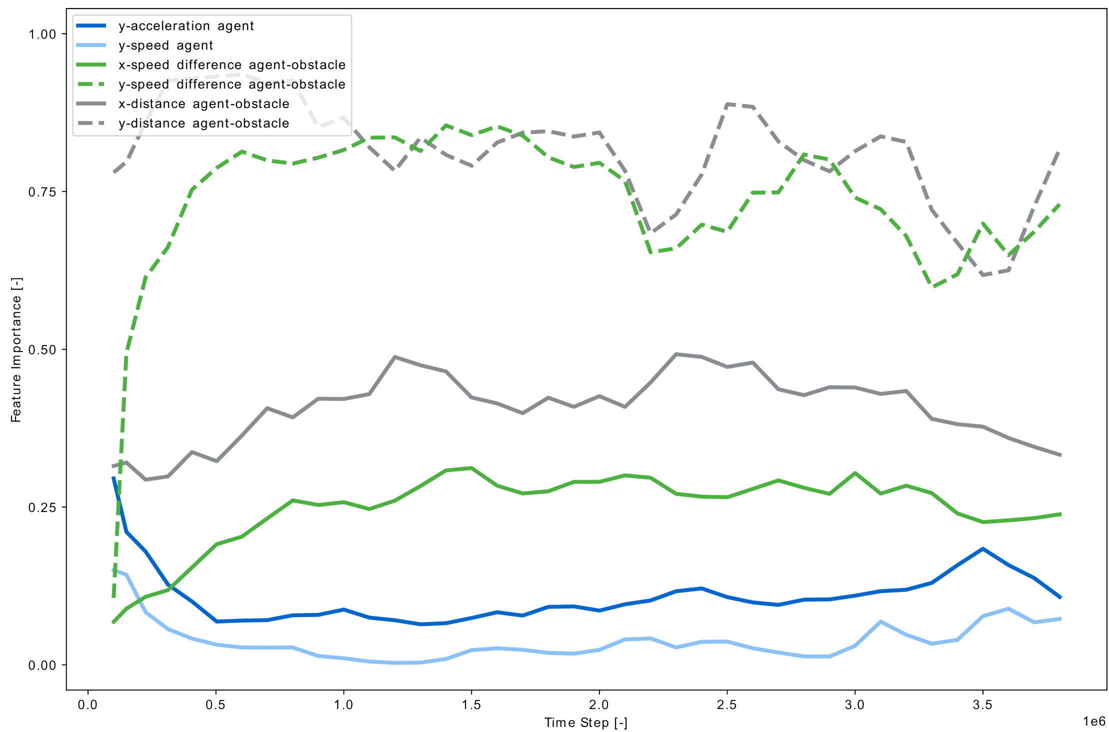

Figure 5.6: SHapley Additive exPlanations (SHAP)-based Feature importance during the training of an agent for 4000000 time steps. The SHAP values were calculated with a frequency of 100000 time steps.

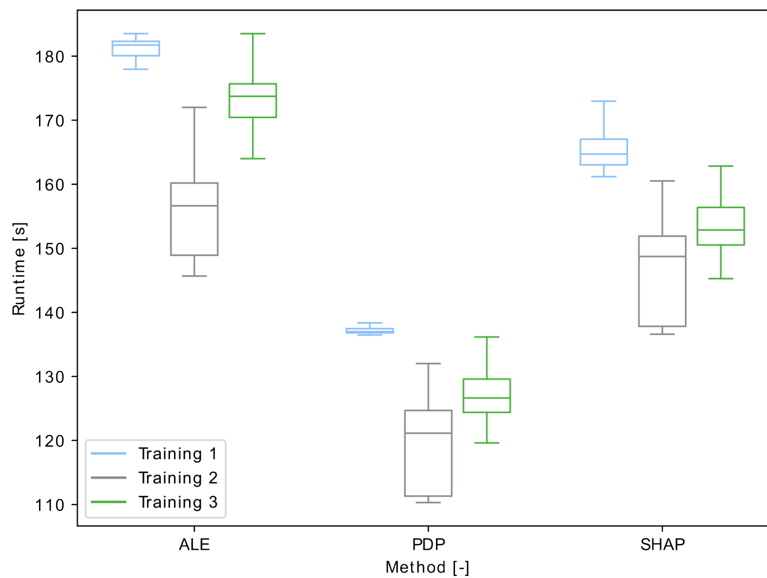

Figure 5.7: Runtimes of the explainability methods. The runtimes were measured at each evaluation step. Three trainings with the same settings but different seed values were executed.

metrics MAE and MSE on the training and test data split are listed in table 5.2.

The metrics MAE and MSE stayed at the same value range or did not change at all for varying depths of the decision tree. Furthermore, the differences between MAE and

Maximal Depth	MAE training [-]	MSE training [-]	MAE test [-]	MSE test [-]
3	0.585	0.445	0.586	0.447
4	0.584	0.445	0.586	0.447
5	0.584	0.444	0.586	0.448
6	0.583	0.443	0.586	0.448

Table 5.2: Training results of the decision trees. The metrics Mean Absolute Error (MAE) and Mean Squared Error (MSE) were evaluated on the training and test data split. The trees vary in maximal depth.

MSE evaluated on the training and test split are marginal. The absolute values of the metrics are large compared to the agent's action range of [-1, 1]. An MAE of 0.585, for example, represents an error of 30 % of the action's total range.

As explained in section 4.3 (Application Settings), the decision tree's performance was further evaluated with newly sampled environment configurations. Two "Simple-OA" environments, described in section 4.1 (Environment Representation), were sampled randomly. The environments are visualized in figure 5.8.

The trained DDPG agent was applied in the environments. For every state the agent encountered, the agent's action computed on that state and the decision tree's output were collected. The agent's actions and the decision tree's outputs were then compared for all time steps the agent needed to finish the episode.

The comparison for the first environment (figure 5.8 (a)) is visualized in figure 5.9 for a decision tree with a maximal depth of four (a) and six (b). During the first half of the time steps, the agent's actions (blue line) had positive values with a maximum at time step 20. Before time step 20, the action values were rising and after they reached a maximum value, they were falling. This trend could be interpreted as the agent was certain about its actions during those time steps. From time step 35 till 50, the actions had no clear trend. After time step 50, the actions fell to the maximal negative value of -1. This could be interpreted as a shift in the agent's decision after time step 50.

Both decision trees produced almost identical values ((a) and (b)). The decision trees' outputs stayed at a similar level for all time steps. From time step 15 till 30, a section of lower values compared to the rest of the values can be identified. However, this does not match with the actions of the agent. Therefore, neither of the decision trees could adapt to the agent's actions in this scenario.

The comparison for the second environment (figure 5.8 (b)) is visualized in figure 5.10 for a decision tree with a maximal depth of four (a) and six (b). The agent's actions (blue line) have clear trends with high absolute values in positive and negative direction. The overall development of the actions appear smoother compared to the actions in the first environment (figure 5.9). This could be interpreted as the agent's

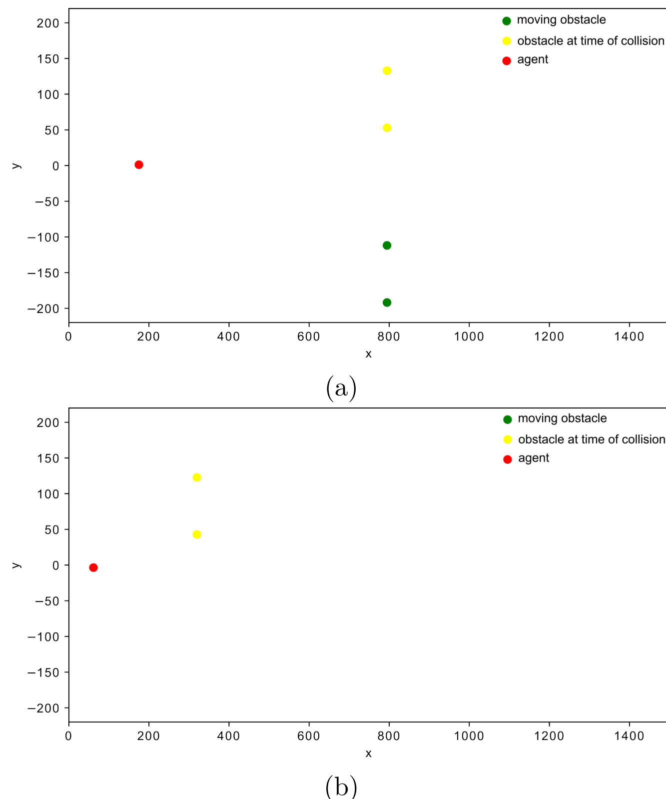

Figure 5.8: Newly sampled "Simple-OA" environments for the decision tree's evaluation. The green dots are the moving obstacles. The yellow dots visualize the position of the obstacles at the time the agent (red) collides with them. Note that in (b) the moving obstacles are outside of the visualization frame at the time the environment was visualized. The trained agent was applied in the environments. For every encountered state, the agent's action and the decision tree's output were collected and compared.

actions having a clear dependency on the states the agent encountered.

The outputs of the decision tree with a maximal depth of four vary in a very small range. Over the whole time span, the values are constantly at a small negative number. From time step 30 till 45, the output values varied in a small range.

The output values of the decision tree with a maximal depth of six vary in a slightly bigger range than the values of the tree with depth three. The variations happened approximately at the same span of time steps for both trees. The values of the tree with depth 6 are positive from time step 30 till 45.

The decision tree with a maximal depth of four could not adapt to the agent's actions in this scenario. Its output values vary in a very small range only. The variation of the tree with a maximal depth of six could be interpreted as a slight adaptation to the agent's actions. The tree computed positive values from time step 30 till 45. This matches the positive peak in the agent's actions at the same time steps. Other variations in the agent's actions could not be reproduced by this tree.

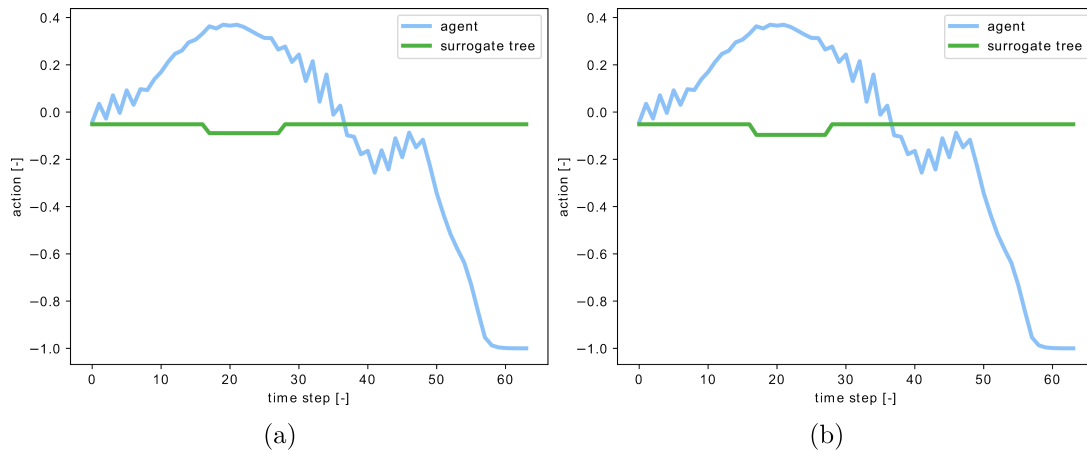

Figure 5.9: Comparison between the agent's actions (blue line) and the decision tree's outputs (green line) in the first sampled environment. The decision tree was trained as a surrogate model on a dataset with the agent's actions as labels. (a) shows the values of a tree with a maximal depth of four and (b) the values of a tree with a maximal depth of six.

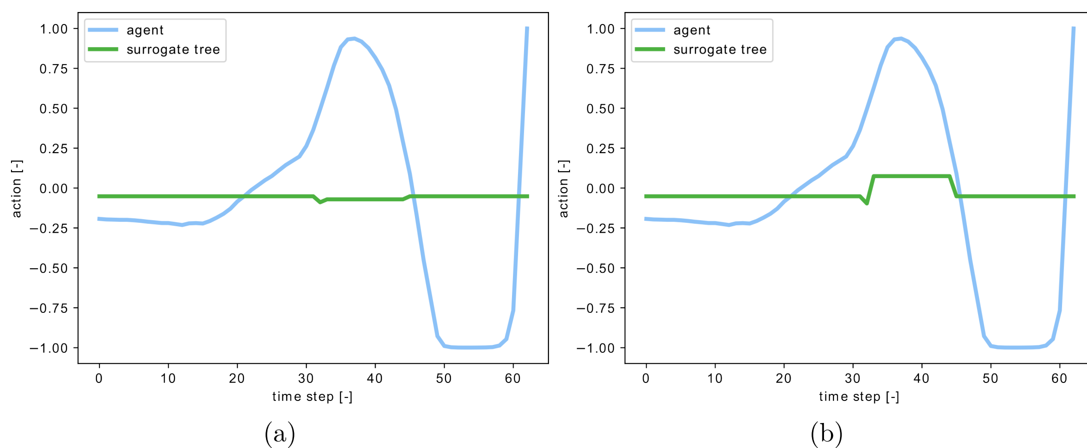

Figure 5.10: Comparison between the agent's actions (blue line) and the decision tree's outputs (green line) in the second sampled environment. The decision tree was trained as a surrogate model on a dataset with the agent's actions as labels. (a) shows the values of a tree with a maximal depth of four and (b) the values of a tree with a maximal depth of six.

6 Discussion

The results of the case study described in chapter 4 (Case Study: Obstacle Avoidance) were presented in chapter 5. The ability of the individual methods to provide explanations about a DRL agent's decision making were investigated. Furthermore, the feasibility in terms of computational overhead were examined.

The results of the method iPDP for the increase of runtime were listed in table 5.1. The application of the iPDP resulted in an increase of runtime of 10.32 hours on average compared to a training process without the application of any explainability method. As mentioned in the corresponding section 5.1 (Problems of the Incremental Partial Dependence Plot), this computational overhead was considered to be too large for the method to be applied in the context of DRL. Therefore, no further investigation of the method iPDP was conducted.

Muschalik et al. mentioned that "computing iPDP for every feature with high fidelity (larger grid sizes) might become infeasible for some application domains" [21]. Even with the low grid resolution of five grid points, computing iPDP for the six features in the state representation led to a large computational overhead.

Additionally, the method was designed to explain ML models that adapt to each new data point. However, the chosen DDPG agent collects state, action and reward tuples over multiple episodes. Only with a chosen frequency, the agent's policy is updated with the collected tuples. Therefore, applying the iPDP at every time step in the training does not account for the method's original purpose.

The results of the methods PDP, ALE and SHAP for explaining the individual features' effect on the agent's actions were presented in section 5.2.1 (Feature Importance). The visualizations of the PDP (figure 5.1) and ALE(figure 5.3) at an exemplary time step provided information about how the individual features influence the agent's decision. The plots of both methods showed similar trends for each feature.

The PDP enables an interpretation of the agent's action dependence on the features averaged over the dataset at a relatively coarse grained grid An advantage over the ALE is its simpler interpretation. The ALE can show a feature's effect on the agent's action at a much finer grid resolution. Furthermore, the ALE can account for possible correlations between the features. The PDP on the other hand, is influenced by correlations between the features. However, the interpretation of the ALE is more difficult resulting from a more complex computation compared to the PDP.

The SHAP dependence plots for the individual features could in addition provide non-averaged explanations. The plots can reveal if a feature's influence on the agent's decision has a clear trend or not. PDP and ALE might result in wrong interpretations if a feature's influence on the action does not have a clear trend.

The resulting plots for feature importance over the agent's training time (figure 5.2, 5.4 and 5.6) enable an overview about the development of the individual feature importances. However, the individual methods' plots should be considered when interpreting the importance of features.

The runtimes of the individual methods were strongly influenced by the chosen application settings. The PDP resulted in the lowest average runtime (figure 5.7). This was achieved by a coarse grained evaluation grid. The average runtime of the ALE was higher since the grid resolution was significantly higher than the one for the PDP.

The runtime of the SHAP method was influenced by a much smaller amount of data points used for the calculation. Whereas all 100000 previously encountered states were used to calculate the PDP and the ALE, only 200 data points randomly sampled from the 100000 states were used to calculate the SHAP values. This setting makes the SHAP method less reliable than the other methods.

A decision tree with different maximal depths was trained as a surrogate model to the RL agent. The resulting values for the metrics MAE and MSE on the training and test data split were listed in table 5.2 in section 5.3 (Global Surrogate Model). The individual values did not vary with the increase of the tree's depth across all metrics. Furthermore, the values of the metrics did not vary either between the tree's evaluation on the training and test data split. In contrast to the range of the agent's action, the values represented an error of 30 % of the total range. According to these results, the decision tree did not adapt properly to the agent's policy.

The comparison of the agent's actions with the decision tree's outputs on newly sampled environment settings showed similar results. In the first sampled environment (figure 5.9), the tree's outputs were almost constant at a value around zero whereas the agent's output varied in positive and negative direction. In the second sampled environment (figure 5.10), the tree with a maximal depth of four computed almost constant outputs again. The tree with depth six showed a variation at a frame of time steps for which the agent's actions had a peak value. This could be interpreted as that the tree adapted to a general trend for the occurred states in these particular time steps.

The described results lead to the interpretation that the decision tree with the chosen depths could not adapt to the agent's policy. However, a close enough adaptation is necessary for a surrogate model to provide explanations about a more complex black-model. Therefore, the decision tree with the chosen hyperparameters can not be used

to interpret the DRL agent used in the case study.

7 Conclusion

The aim of this work was to investigate and evaluate the application of explainability methods in RL scenarios. The first part of the work investigated the application of explainability methods during an agent's training process. Therefore, the methods PDP, ALE and SHAP were chosen. The second part of the work investigated the ability of a decision tree to function as a surrogate model for a trained agent.

In the first part, the chosen explainability methods were applied in a case study. In an environment consisting of two moving obstacles, a DDPG agent was trained to pass in between the two obstacles. This scenario was interpreted as a "moving finish line". The agent was trained for 4000000 time steps and the training was repeated three times. During the training, the explainability methods were applied with a frequency of 100000 times steps.

In the second part, a decision tree was applied as a surrogate model to the agent. Therefore, the last 100000 states the agent encountered during the training were used as the training dataset. The trained agent's predictions on these states were used as labels in the training of the decision tree.

The methods applied during the training were able to provide explanations about the individual features' influence on the agent's actions. The exemplary plots of PDP and ALE showed similar trends for the influence of the features on the agent's actions. While the PDP was evaluated on five grid points, the ALE was evaluated with a much finer grid resolution. The SHAP dependence plots showed similar trends for the influence of the features on the agent's actions for some of the features. The SHAP values of the other features had no clear trend. The computational overhead of the different methods varied in a small range resulting from the individual application settings of the methods.

Concluding from this, all three methods could add explanations about the influence of the features on the agent's actions. The PDP adds plots with a simple and clear interpretation. However, the grid resolution had to be small to keep the computational overhead at a low range. The ALE could be computed at a much higher grid resolution with a computational overhead in the same range as the PDP. Therefore, the influence of the individual features are visualized more detailed. Additionally, the ALE method accounts for correlations between features while the PDP is influenced by correlations. The SHAP method added more detailed insights about the influence of the individual features than the previous methods. Instead of averages, values for individual data points are visualized. This can reveal if trends found by the previous methods are

actually present or if they resulted from averaging over predictions without any trend. However, as a result of a large computational overhead, the SHAP method was calculated only for 200 data points instead of 100000 data points used by the other methods. Therefore, the results of the SHAP method might be less reliable than the results of PDP and ALE.

The decision tree applied as a surrogate model could not adapt to the agent's policy. The evaluated metrics on the training and test data split as well as the comparison between the tree's outputs and the agent's actions showed that the tree could not provide explanations about the policy.

In further investigations, the decision tree could be replaced by a different model as a surrogate model for the agent. For example, Gjærum et al. [7] applied a linear model tree which led to a better adaptation of an RL agent's policy. However, the ability of a model to work as a surrogate model highly depends on the RL scenario and the trained agent.

In conclusion, the agent in the presented case study could benefit from a different surrogate model which can adapt more closely to the agent's policy. The chosen explainability methods that were applied during the training could provide valuable explanations through visualizations of the individual features' influence on the agent's actions. This could help developers improve the performances of DRL agents.

Bibliography

[1] Adadi, Amina and Mohammed Berrada: *Peeking inside the black-box: a survey on explainable artificial intelligence (XAI)*. IEEE access, 6:52138–52160, 2018.

[2] Apley, Daniel W and Jingyu Zhu: *Visualizing the effects of predictor variables in black box supervised learning models*. Journal of the Royal Statistical Society Series B: Statistical Methodology, 82(4):1059–1086, 2020.

[3] Bellman, Richard: *The theory of dynamic programming*. Bulletin of the American Mathematical Society, 60(6):503–515, 1954.

[4] Emerj, the AI Research and Advisory Company: *What is machine learning?* https://emerj.com/ai-glossary-terms/what-is-machine-learning/. Accessed: March 29, 2024.

[5] Friedman, Jerome H: *Greedy function approximation: a gradient boosting machine*. Annals of statistics, pages 1189–1232, 2001.

[6] Géron, Aurélien: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[7] Gjærum, Vilde B, Inga Strümke, Jakob Løver, Timothy Miller and Anastasios M Lekkas: *Model tree methods for explaining deep reinforcement learning agents in real-time robotic applications*. Neurocomputing, 515:133–144, 2023.

[8] Greenwell, Brandon M, Bradley C Boehmke and Andrew J McCarthy: *A simple and effective model-based variable importance measure*. arXiv preprint arXiv:1805.04755, 2018.

[9] Hart, Fabian, Martin Waltz and Ostap Okhrin: *Missing Velocity in Dynamic Obstacle Avoidance based on Deep Reinforcement Learning*. arXiv preprint arXiv:2112.12465, 2021.

[10] Heuillet, Alexandre, Fabien Couthouis and Natalia Díaz-Rodríguez: *Explainability in deep reinforcement learning*. Knowledge-Based Systems, 214:106685, 2021.

[11] Hu, Yuh-Jong and Shang-Jen Lin: *Deep reinforcement learning for optimizing finance portfolio management.* In *2019 amity international conference on artificial intelligence (AICAI)*, pages 14–20. IEEE, 2019.

[12] Jain, Anil K, Jianchang Mao and K Moidin Mohiuddin: *Artificial neural networks: A tutorial.* Computer, 29(3):31–44, 1996.

[13] Kiran, B Ravi, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani and Patrick Pérez: *Deep reinforcement learning for autonomous driving: A survey.* IEEE Transactions on Intelligent Transportation Systems, 23(6):4909–4926, 2021.

[14] Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver and Daan Wierstra: *Continuous control with deep reinforcement learning.* arXiv preprint arXiv:1509.02971, 2015.

[15] Lundberg, Scott M, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal and Su-In Lee: *From local explanations to global understanding with explainable AI for trees.* Nature machine intelligence, 2(1):56–67, 2020.

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

[17] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski et al.: *Human-level control through deep reinforcement learning.* nature, 518(7540):529–533, 2015.

[18] Molnar, Christoph: *Interpretable Machine Learning.* 2 edition, 2022.

[19] Morales, Eduardo F, Rafael Murrieta-Cid, Israel Becerra and Marco A Esquivel-Basaldua: *A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning.* Intelligent Service Robotics, 14(5):773–805, 2021.

[20] Murdoch, W James, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl and Bin Yu: *Definitions, methods, and applications in interpretable machine learning.* Proceedings of the National Academy of Sciences, 116(44):22071–22080, 2019.

[21] Muschalik, Maximilian, Fabian Fumagalli, Rohit Jagtani, Barbara Hammer and Eyke Hüllermeier: *iPDP: On Partial Dependence Plots in Dynamic Modeling Scenarios.* In *World Conference on Explainable Artificial Intelligence*, pages 177–194. Springer, 2023.

[22] Plaat, Aske: *Deep reinforcement learning.* Springer, 2022.

[23] Shapley, Lloyd S et al.: *A value for n-person games.* 1953.

[24] Sutton, Richard S. and Andrew G. Barto: *Reinforcement Learning: An Introduction.* MIT Press, 2018.

Appendix A

Incremental Partial Dependence Results

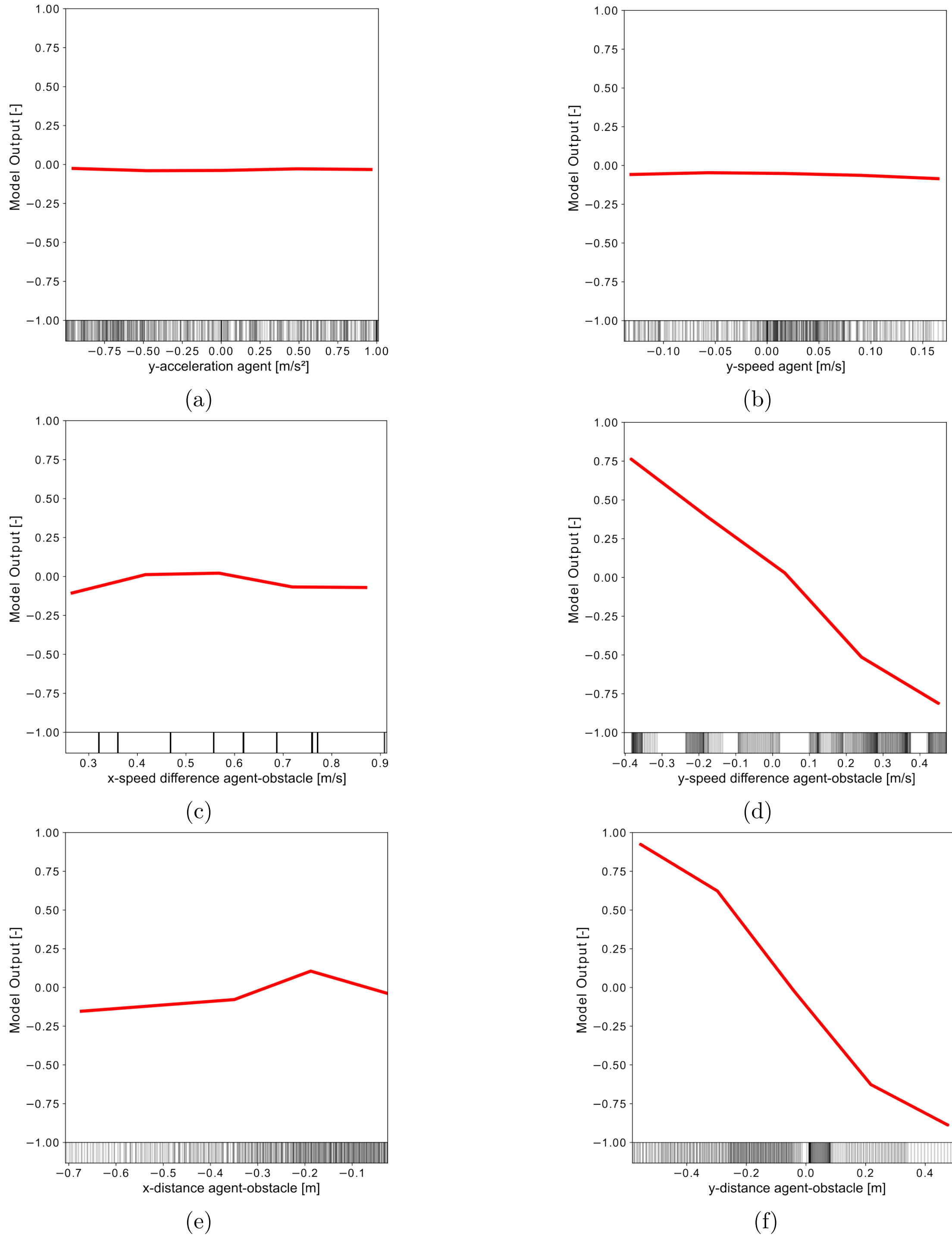

Figure A.1: Individual incremental Partial Dependence Plots (iPDPs) for the six features in the state representation. The vertical lines at the bottom of each plot visualize the distribution of the corresponding feature in the data set. 500 data points are visualized. According to the iPDPs, Features (d) "y-speed difference agent-obstacle" and (f) "y-distance agent-obstacle" have the biggest influence on the model output.

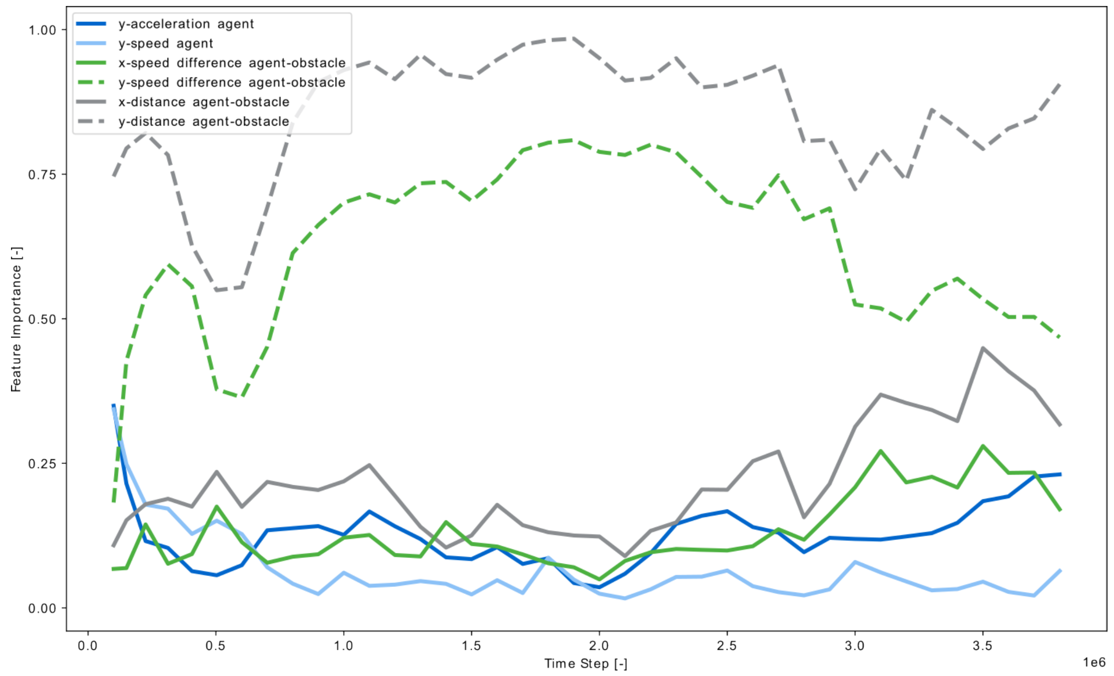

Figure A.2: Feature importance based on the incremental Partial Dependence Plot (iPDP). The individual lines represent the change of the corresponding feature importance during the training of a reinforcement learning agent for 4000000 time steps. The Partial Dependence Plot (PDP)-based feature importance was used to calculate the values [8]. The values are exponentially smoothed with $\alpha = 0.5$.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus fremden Quellen wörtlich oder sinngemäß übernommenen Gedanken sind als solche kenntlich gemacht. Ich erkläre ferner, dass ich die vorliegende Arbeit an keiner anderen Stelle als Prüfungsarbeit eingereicht habe oder einreichen werde.

Dresden, August 26, 2024

Jonas Keller