Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

David Aleksander Kinn

Causal Episode Explanations for Reinforcement Learning Applications

Master's thesis in Cybernetics and Robotics (Master, 2 years)
Supervisor: Anastasios Lekkas
Co-supervisor: Sindre Benjamin Remman
June 2023

NTNU
Norwegian University of
Science and Technology

David Aleksander Kinn

Causal Episode Explanations for Reinforcement Learning Applications

Master's thesis in Cybernetics and Robotics (Master, 2 years)
Supervisor: Anastasios Lekkas
Co-supervisor: Sindre Benjamin Remman
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Preface

I have written my master's thesis on the topic of Explainable Arteficial Intelligens (XAI), which is completed on behalf of the Norwegian University of Science and Technology. The field of XAI is both fascinating and challenging, and I am proud to have had the opportunity to delve into it under the guidance of my supervisor, Anastasios Lekkas, and my co-supervisor, Sindre Benjamin Remman.

My motivation for pursuing this topic stems from the realization that while there are several methods available for explaining Arteficial Intelligens (AI) models, there is a dearth of techniques that specifically address the explainability of Reinforcement Learning (RL) applications. As someone who is passionate about RL and its potential to drive innovation, I was motivated to explore how to explain RL applications sufficiently and contribute to my field of expertise.

Through my project thesis and AI classes in my master's program, I gained in-depth knowledge of both RL and XAI, which prepared me well for my master's thesis. However, the main challenge in my thesis was to translate my conceptual idea into a tangible implementation. In particular, designing my own optimization objective and solver, specializing in making the explanation graph sparse, proved to be a challenging yet rewarding experience.

I take pride in the progress I have made, and I am pleased to present a new technique in my thesis. This technique, which I hope readers will find interesting, addresses the issue of explaining RL applications and has the potential to contribute to ongoing research in the field.

I am grateful to my supervisors, Anastasios Lekkas and Sindre Benjamin Remman, for their support and guidance throughout my thesis. Their encouragement helped me overcome my challenges and inspired me to push my boundaries to achieve my goals.

Abstract

This thesis presents a new method for comprehensively explaining how a Reinforcement Learning agent acts in an episode. The method explains why the agent takes its actions and how these actions affect its future states. We refer to these as the causes and effects of the agent, respectively, which is why we call the explanations Cause and Effect Sequential (CES) explanations. CES explanations are kept simple by grouping similar subsequent actions and limiting the states and actions mentioned in the explanation to the most influential ones. The results of the method indicate that the method generally works well. Assessments taken by a well-trained agent seem logical, while the evaluations are more illogical for a poorly trained agent (as they should be). We validate parts of the explanation with SHapley Additive exPlanations (SHAP) and conclude that the methods usually agree. The author suggests improving the method further by adjusting the action grouping parameters automatically, allowing for linear increasing or decreasing action groups, and doing several measures to find the influential parts of the episodes more robustly.

Sammendrag

Denne master oppgaven presenterer en ny metode for omfattende forklaring av hvordan en forsterkningslærende agent handler i en episode. Metoden forklarer hvorfor agenten tar sine handlinger og hvordan disse handlingene påvirker dens fremtidige tilstander. Vi refererer til disse som årsakene og virkningene av agenten, henholdsvis, og er derfor vi kaller forklaringene for Cause and Effect Sequential (CES) forklaringer. CES forklaringer holdes enkle ved å gruppere lignende påfølgende handlinger og begrense tilstandene og handlingene nevnt i forklaringen til de mest innflytelsesrike. Resultatene fra metoden indikerer at metoden generelt fungerer bra. Evalueringer gjort av en godt trent agent virker logiske, mens evalueringene tatt av en dårlig trent agent er vanligvis er mer ulogiske (som de bør være). Vi validerer deler av forklaringen med SHapley Additive exPlanations (SHAP) og konkluderer med at metodene vanligvis er enige. Forfatteren foreslår å forbedre metoden ytterligere ved å justere parameterne for handlinggruppering automatisk, tillate lineær økning eller reduksjon av handlinggruppene, og å gjøre flere tiltak for å robust finne de innflytelsesrike delene av episodene.

Table of Contents

List of Figures

List of Tables

List of Acronyms

CHAPTER

Introduction 1

Arteficial Intelligens (AI) has advanced rapidly in recent years, transforming numerous industries and permeating our daily lives. From marketing to education, AI has the potential to revolutionize how we operate and enhance our decision-making capabilities. However, as we increasingly rely on AI to make critical decisions, there are growing concerns over the need for more transparency and interpretability of AI models. Thus, the need for Explainable Arteficial Intelligens (XAI) has become increasingly important. To make the decision-making explainable, XAI research either proposes making interpretable models to begin with or drawing simplified explanations on how the model works.

While XAI research has made significant progress in recent years, there is still limited work on applying XAI to Reinforcement Learning. As ChatGPT uses Reinforcement Learning as part of its training [21], matrix multiplication was optimized using Reinforcement Learning [8], and it exists Reinforcement Learning agents that are almost unbeatable for humans in chess and Go [2]; Reinforcement Learning indeed has great potential.

Methods such as Counterfactuals [16], SHapley Additive exPlanations (SHAP) [25], and Local Interpretable Model-Agnostic Explanations (LIME) [36] aim to make black box models more interpretable but are generally more suited to explain supervised learning problems than explaining Reinforcement Learning. Although these methods can explain each iteration of an agent separately, they will not provide comprehensive assessments of episodes. In physics-based environments, each iteration in itself often has a relatively low effect on the complete episode. Furthermore, comprehensive assessments are necessary to understand what is happening.

To address this gap in research, this article will introduce a new method called Cause and Effect Sequential (CES) explanation. It aims to post-evaluate and explain an episode for an agent and an environment with non-interpretable relationships between its state transitions. States in the context of Reinforcement Learning mean all the relevant information the agent needs to make its decision. The method groups the actions, studies which actions influence the agent most to end up in its final state, and investigates which states the agent considers the most. This will strengthen our trust in the agent and give the user a better understanding of how each state and action are related.

The method allows and aims explicitly to explain multidimensional and continuous action and state spaces. This makes the method great for explaining control applications for systems such as drones or vessels, as they operate in continuous state spaces. Therefore, in this thesis, we will evaluate the method's performance by explaining the interaction of a Proximal Policy Optimization (PPO) agent on the widely used Lunar Lander environment.

CHAPTER

Theory 2

The Theory chapter in this work is structured to provide the necessary background and knowledge required to understand how CES explanation works. Naturally, the reader needs to understand the fundamentals of Reinforcement Learning and have an insight into existing XAI approaches. Since CES explanations have closely related elements to the XAI method SHAP, we will detail SHAP's functioning. We will also present causality, causal models (graphs), and Pearls do-calculus and relate Pearls do-calculus to SHAP. Finally, we will present the key ideas behind global optimization since CES explanations use multiple objective functions and solvers.

The thesis aims for an audience with a good understanding of how neural networks work. Neural network fundamentals will, therefore, not be covered.

2.1 Reinforcement Learning

Besides supervised and unsupervised learning, there is a distinct category of AI called Reinforcement Learning. What distinguishes Reinforcement Learning from Supervised Learning is that we can not know the model's performance based on one output. In a reinforcement learning problem, the decision-making model called the agent, tries to achieve its goal by taking multiple actions.

The state orients the agent about how close it is to achieving its goal. Hence for every action that gets the agent closer or further away from its goal, the state will change accordingly. These state transitions are determined by what is called the Environment. Figure 2.1 provides an overview of how these components work together. Here s_t is the state, a_t is the action, and R_t is the reward at time t. We will go further into the reward in the upcoming subsection.

The author aims to prevent the reader from believing that Reinforcement Learning has a narrow range of applications. In reality, plenty of problems can be formed as Reinforcement Learning problems. Here are just a few examples:

1. In ChatGPT, they use something called Reinforcement Learning from Human Feedback (RLHF) to train the model [21]. In language models such as ChatGPT, the state can be defined as the text generated by the model and the context in which it is generated (e.g., a prompted question from the human). The action can be the next generated word in the sentence.
2. A Reinforcement Learning agent can master games such as chess. Here the states can be the position of the pieces, and the action can be the player's next move [2].
3. Reinforcement Learning can be used in control applications. If we want to control a vessel, we can define the states as its kinematics and kinetics and the actions the control inputs to its actuators.
4. DeepMind has improved matrix multiplication with Reinforcement Learning [8]. Increasing the speed of matrix multiplications will increase the speed of GPU-intense processes in machine learning, physics simulations, gaming, and more. Here the 2D multiplication is converted into a 3D tensor, and our goal is to find the least number of decomposed vector products that sum up to our tensor [8]. The state is the tensor minus the decomposed vector products yet produced, and the action is to find new decomposed vectors [8].

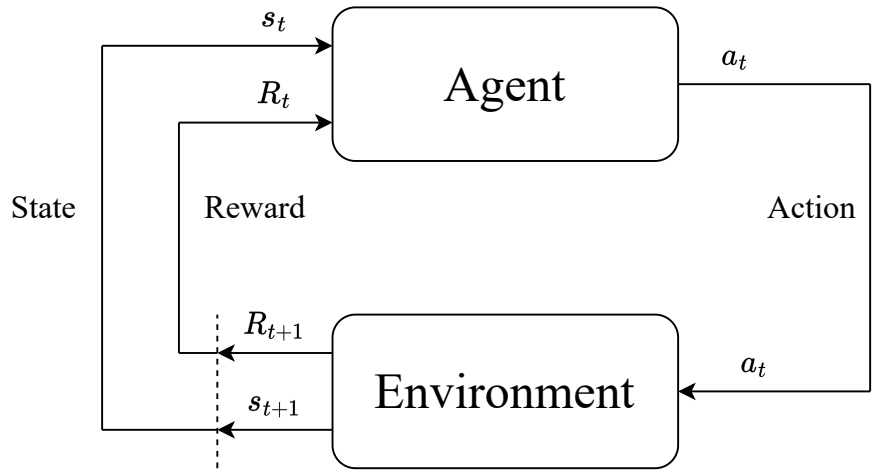


Figure 2.1: A basic illustration of how an agent interacts with an environment in a Reinforcement Learning application.

2.1.1 Learning

Reinforcement Learning is a method of learning that involves reinforcing desired behaviors through rewards. A reward function is introduced to distinguish which actions had the highest degree of influence in achieving the agent's goals. The reward function is application dependent, and its design can not be generalized. Having said that, it is advantageous that the reward function is continuous, not sparse [7]. Continuously updating feedback from the reward makes it generally easier for the agent to distinguish its good and bad actions [7].

A real-life example of Reinforcement Learning is a dog owner learning a dog new tricks. When the dog owner wants to learn a dog "roll over," he/she rewards the dog when it lies down since it is one step closer to understanding the tricks.

Reinforcement Learning applications are partially constrained due to learning challenges. Learning often becomes unstable and has a problem converging due to its indirect training approach [27]. We want the agent to follow a policy that maximizes its current and future rewards, and future rewards are challenging to estimate. The upcoming sections go into more theoretical details of Reinforcement Learning implementations and how they work. Subsection 2.1.2 to 2.1.7 is almost taken directly from the project thesis.

2.1.2 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical framework for modelling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker [31]. It provides a way to represent and solve problems in which an agent interacts with an environment, trying to maximize some reward.

An MDP consists of a set of states, actions, and a transition function that defines the probability of transitioning from one state to another as a result of taking a particular action. It also includes a reward function that assigns a numerical reward or penalty to each state-action pair. The goal of an MDP is to find a policy that maximizes the expected cumulative reward over time.

One noteworthy aspect of MDPs is the concept of temporal discounting, which allows the model to consider the value of future rewards relative to the present [31]. Temporal discounting is useful when the agent has to trade off immediate rewards for longer-term benefits.

MPDs have real-world use cases outside the Reinforcement Learning fields, such as Economic Quality control [23].

Although some processes are not random, randomness compels exploration. By encouraging exploration, we get a reasonable estimate and avoid making greedy, uneducated choices. Equation 2.1 shows the probability of a certain state transition based on the MDP and Equation 2.2 shows the objective to maximize.

$$P_{\boldsymbol{a}}(\boldsymbol{s}, \boldsymbol{s}') = \Pr(\, \boldsymbol{s}_{t+1} = \boldsymbol{s}' \mid \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}) \tag{2.1}$$

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{\boldsymbol{a}_t}(\boldsymbol{s}_t, \boldsymbol{s}_{t+1})\right] \tag{2.2}$$

2.1.3 The Bellman equation

The Bellman equation is a central concept in Markov Decision Processes (MDPs) theory. Given the MDP's current policy and transition probabilities, it provides a way to compute how good a state is for us to maximize the current and future rewards [31]. Equation 2.3 shows a general structure of the Bellman equation. The Equation has reward $R(s, a)$ and a discount factor γ to prioritize the state of the next iteration over the more uncertain future reward outcomes.

$$V(s) = \max_{\boldsymbol{a}}(R(\boldsymbol{s}, \boldsymbol{a}) + \gamma V(\boldsymbol{s}')) \tag{2.3}$$

Often we extend this as a probabilistic equation as in Equation 2.4. An arrow is used instead of an equal sign because it is an update of the value and may vary per iteration.

$$V(s) \leftarrow \max_{\boldsymbol{a}} \sum_{\boldsymbol{s}'} p\,(\, \boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}\,)\,[R(\boldsymbol{s}, \boldsymbol{a}) + \gamma V(\boldsymbol{s}')] \tag{2.4}$$

2.1.4 Q-value and policy

Now let us extend our knowledge by introducing two new concepts: Q-value and policy. Q-value is very similar to the Bellman equation return $V(s)$, with the crucial difference that it varies with the action. Equation 2.5 shows the relationship between the optimal value function and Q-function, and Equation 2.6 shows an update rule for the Q-value [31]. α, often referred to as the step size, decides how much effect the new iteration has on the updated Q-value. The updated rule here is very similar to the $V(s)$ with one essential difference: there is no need to sum up the states. Dealing with Q-values is often beneficial in continuous state spaces, so we do not need to discretize the states.

$$V_*(s) = \max_{\boldsymbol{a}} Q_*(\boldsymbol{s}, \boldsymbol{a}) \tag{2.5}$$

$$Q(\boldsymbol{s}, \boldsymbol{a}) \leftarrow Q(\boldsymbol{s}, \boldsymbol{a}) + \alpha \left(\max_{\boldsymbol{a}'} [R(\boldsymbol{s}, \boldsymbol{a}) + \gamma Q(\boldsymbol{s}', \boldsymbol{a}')] - Q(\boldsymbol{s}, \boldsymbol{a})\right) \tag{2.6}$$

$V(s)$ and $Q(s, a)$ uses the max argument, which requires a discrete action space to compare all actions. For continuous action spaces, it would be very beneficial to have a function that approximates the optimal policy for a given state. See Equation 2.7 for the optimal policy we want to approximate [31].

$$\pi_*(s) = \arg\max_{\boldsymbol{a}} \sum_{\boldsymbol{s}'} p\,(\, \boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}\,)\,[R(\boldsymbol{s}, \boldsymbol{a}) + \gamma V_*(\boldsymbol{s}')] \tag{2.7}$$

2.1.5 Network approximation

With limited action and state spaces, finding good or optimal policy, value function, and Q-function would be possible by exploring all state-action pairs. In large or continuous spaces, this will be time-consuming or even impossible. We, therefore, need a way to approximate untried state-action pairs based on the causal relationships between state space, action space, and rewards.

Neural networks make this possible, where we can represent the Q-function, the value function, and the policy as a neural network. Equations 2.4 and 2.6 motivates the elements of the loss functions for the Q- and value- networks, respectively. If we were to base our policy network on Equation 2.7, the loss of a policy network either depends on a Q estimate or a value estimate. This makes such a tuning process trickier. Section 2.1.7 shows a way to tackle this problem.

2.1.6 On- versus off-policy Learning

On- and Off-Policy Learning are methods defining how the agent learns from the environment. In On-Policy Learning, the agent independently chooses how to react with the environment; the agent plays its episodes and learns from the reward. The agent tries taking the actions believed to be best for every iteration.

On the other hand, off-policy networks can learn from the data generated by an arbitrary action-maker. The agent is not necessarily responsible for the generated data, although it typically learns from its previous episodes. Here the agent learns from recorded data, often presented in random order. It is, therefore, usually tuned using replay buffers with self-defined batch sizes. The data is reusable, leading to more sample efficiency.

2.1.7 Actor-Critic

Finding a good policy has its difficulties. The actor-critic method combines value- and policy-based networks to tackle those problems, allowing to solve RL problems with continuous state and action spaces. First, the actor uses a policy network to choose an action. Next, the Critic will evaluate how good of an action it was, for example, using a Q-network. This way, we get the best of both worlds, which could be why we see new actor-critic methods relatively often.

2.2 Explainable Artificial Intelligence (XAI)

XAI refers to the ability to provide understandable explanations for Artificial Intelligence's decision-making processes. In other words, XAI aims to make AI more transparent and interpretable so humans can understand why and how an AI system arrives at a particular decision or recommendation. This is especially important in high-stakes applications. An example of a high-stake application that is a Reinforcement Learning problem is the control of autonomous vessels.

The need for XAI arises because many AI systems, especially those based on neural networks, have internals that are overwhelming to interpret. While these systems, often described as black-box models, can make complex decisions with high accuracy, the lack of transparency can make it difficult to trust them and identify potential biases or errors.

We can divide the explanation methods into Transparent models and Post-Hoc techniques [29]. Transparent models are machine learning models specifically designed to have an interpretable structure. This way, we do not need any top-layer techniques to provide explanations. A drawback of taking this approach is that the complexity of the model is restricted to what a human can comprehend.

A Post-Hoc technique works on top of a black box model and simplifies its decision-making when producing the explanation. Post-Hoc techniques are divided into model-agnostic methods, such as

LIME and SHAP, and model-specific methods, such as gradient-based Counterfactual explanations [29]. Model-agnostic methods aim to explain any machine learning model, whereas model-specific methods are tailored to specific models [29]. A model-specific method can, for example, have access to the gradient of a neural network and base its explanation on that.

There is also a distinction between the scope of the explanation. Some methods try to explain global attributes to the model [29]. At the same time, some techniques take a more local approach, e.g., explaining a specific output by inspecting the model properties in the local area. Permutation Feature Importance is an example of a global technique, while LIME is an example of a Local technique [29].

We will explain three popular Post-Hock methods to give the reader an overview of XAI approaches. In this section, we will describe LIME and Counterfactual explanations and continue by explaining SHAP. SHAP deserves its own subsection because it shares a similar value function with CES explanations.

LIME is a popular model-agnostic method that generates explanations by approximating a complex model with a simpler, interpretable one [36]. LIME works by perturbing the input data around the point of interest and generating a set of local linear models to approximate the behavior of the complex model in that region [36]. The linear model is evaluated using an optimization objective consisting of a summation of the two functions $\mathcal{L}$ and Ω [36]. $\mathcal{L}$ penalizes the objective based on how inaccurate the linear model is, while Ω penalizes the linear model for its complexity [36]. Hence, a low objective value corresponds to an accurate yet simple explanation.

Counterfactual explanations are formed differently. Here we want an explanation in the form: "If x were equal to x', then y would be c." [16] The method tries to find the smallest and simplest change of input that would result in another output [16]. Accessing the gradient of the input would, therefore, increase the performance, although model-agnostic counterfactual methods also exist. In counterfactuals, we would typically minimize a distance function $d(x, x')$ such that $f(x') = c$ is followed [16]. $d(x, x')$ returns a scalar representing the distance between the original input x and the altered input x', while $f(x') = c$ constraints the model f's output to be our desired output c. The distance function is often designed to alter as few inputs as possible and, in this way, keep the explanations simple. Counterfactuals are also used for adversarial attacks on neural networks [13]. The method can, for example, add a low amount of "strategic" noise to an image of a cat, making an image classifier believe that it is a dog.

2.2.1 SHapley Additive exPlanations (SHAP)

SHAP is a unified framework for interpreting the output of any machine learning model. It is based on the concept of Shapley values from cooperative game theory, which measures the contribution of each player for a given game result [25]. In the context of machine learning, the game is the prediction task, and the players are the input features. The Shapley value of a feature measures how much it contributes to the predicted output, on average, across all possible combinations of features. This makes it a valuable tool for understanding and debugging machine learning models and building trust and transparency in machine learning applications.

The value function in SHAP helps us find how much a subset of the input features contributes to the abnormality of the output. E.g., if an AI model predicts that the life span of an individual is ten years shorter than average, the combination of high BMI and inherited heart decides conditions could be a reason. Then since the life expectancy of the individual is lower than expected, the subset of these two features will result in a negative value function. The formula for the value function is given in Equation 2.8. Here f is the model function, and $f_S(\boldsymbol{x}_s)$ is the expected output of random variable X with overwritten values of input $X_i = x_i$ for all $i \in S$. More detail on $f_S(\boldsymbol{x}_s)$ will be provided in subsection 2.3.1.

$$\upsilon(S) = f_S(\boldsymbol{x}_s) - \mathbb{E}\left[f(\boldsymbol{X})\right] \tag{2.8}$$

SHAP's approach to finding each feature's individual contribution is to take a weighted average of

Symbol	Description
ϕ_i	SHAP value for feature i
F	A set of all the features
S	A subset of the features
f	The function for the black-box model.
x	The feature input we are interested to explain.

Table 2.1: Parameters and description for SHAP value calculation

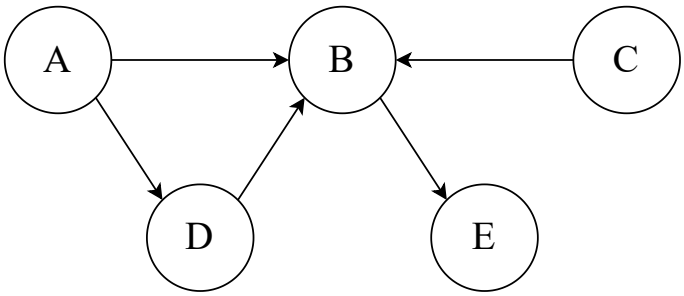


Figure 2.2: Example of a graph of a causal model (a causal graph)

all value functions of subsets including the feature, minus all subsets excluding it. Each subset has a weight based on its number of order-permutations. We will not go into further details since this weight is not directly associated with the thesis, and a thorough explanation would take several pages. For readers that would like to know more about this weight, they can read [4]. Equation 2.9 shows the formula for finding the SHAP values, and Table 2.1 gives the parameter definitions.

$$\begin{aligned}\phi_i &= \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \left[\upsilon(S \cup \{i\}) - \upsilon(S)\right] \\ &= \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \left[f_{S\cup\{i\}}(\boldsymbol{x}_{S\cup\{i\}}) - f_S(\boldsymbol{x}_S)\right]\end{aligned} \tag{2.9}$$

2.3 Causality

In mathematics, causality is a concept that refers to the relationship between cause and effect [33]. It is a fundamental concept used in various mathematics fields, including probability theory and machine learning. The concept of causality is crucial in understanding how different variables affect one another and how we can use this information to make predictions and make informed decisions.

Causality refers to the idea that a certain action or event leads to a particular outcome or effect [33]. Hence, in causality, we may have a relationship between an independent variable (the cause) and a dependent variable (the effect), where changes in the independent variable directly affect the dependent variable.

One of the most common ways of studying causality in mathematics is through the use of causal models. A causal model is a graphical representation of the causal relationships between different variables in a system [33]. The model typically consists of a set of nodes representing different variables and a set of edges representing the causal relationships between them. Figure 2.2 shows an example of a causal model.

Using causal models allows us to identify causal relationships between different variables in a system, even when these relationships are not directly observable. For example, in a medical

study, we may be interested in identifying the causal relationship between a particular treatment and its effect on patient outcomes. By constructing a causal model, we can identify the different variables that are likely to have confounding biases and control these when analyzing the data. Confounding bias occurs when two variables are affected by a common variable [1]. Then the two variables correlate even if they are independent of each other.

In statistics, causality is often studied through the use of randomized controlled trials (RCTs). RCTs are experiments in which subjects are randomly assigned to different treatment groups, with one group receiving the treatment under study and the other group receiving a placebo or an alternative treatment [1]. The study aims to determine whether the treatment has a causal effect on the outcome of interest while controlling for other factors that may affect the outcome.

The use of RCTs allows researchers to establish causality with a high degree of confidence, as the random assignment of subjects to treatment groups ensures that any observed differences between the groups are due to the treatment and not to other factors. However, RCTs can be expensive and time-consuming and may not be feasible or ethical in all situations [35] [34].

In machine learning, causality is becoming an increasingly important area of research, particularly in the context of causal inference and causal discovery. Causal inference refers to studying possible effects by altering the data for a given system [6]. In contrast, causal discovery refers to the process of automatically identifying causal relationships from observational data [15].

2.3.1 Pearl's do-calculus

Pearl's do-calculus is a powerful framework for causal inference that was developed by Judea Pearl, a computer scientist, and philosopher known for his work on causality and Bayesian networks. The do-calculus provides a set of rules for reasoning about causality, allowing us to infer causal relationships between variables from observational and interventional data [32].

The do-calculus is based on interventions, which involve manipulating a variable in a system to observe the effect on other variables. Interventions are distinct from observations, which involve simply measuring the values of variables in a system without manipulating them. The benefit of do-calculus is that interventions can be used to identify causal relationships between variables, even when observational data are insufficient [32].

The key characteristic of do-calculus is the do-operator, denoted as $do(X = x)$. Here X is a random variable that we intervene to be equal to x. In this way, we remove X's statistical dependencies [32].

Within the context of SHAP, the algorithm strongly depends on the do-operator functioning. Earlier, we briefly explained the function $f_S(\boldsymbol{x}_S)$ from Equation 2.8. The more mathematical way to explain the function is $f_S(\boldsymbol{x}_S) = \mathbb{E}\left[f(\boldsymbol{X}) \mid do(\boldsymbol{X}_S = \boldsymbol{x}_S)\right]$. As we can see, we find the expected value of the multidimensional random variable X, except that we overwrite X's dimensions in the subset S with the input instance we are interested in explaining.

Because of its properties, the do-operator provided a new way of finding the Average Treatment Effect (ATE) than with Randomized control trials (RCT). The ATE is calculated for Randomized Controlled Trials by conditioning a variable X to see the effect on output Y. See Equation 2.10. This approach can lead to a confounding result, as we will exemplify in the upcoming paragraph. By conditioning with the do-operator as in Equation 2.11, we exclude all observed and unobserved confounding variables. We see that this way of calculating ATE has many similarities to SHAP's value function.

$$ATE_{RTC} = \mathbb{E}\left[Y \mid X = 1\right] - \mathbb{E}\left[Y \mid X = 0\right] \tag{2.10}$$

$$ATE = \mathbb{E}\left[Y \mid do(X = 1)\right] - \mathbb{E}\left[Y \mid do(X = 0)\right] \tag{2.11}$$

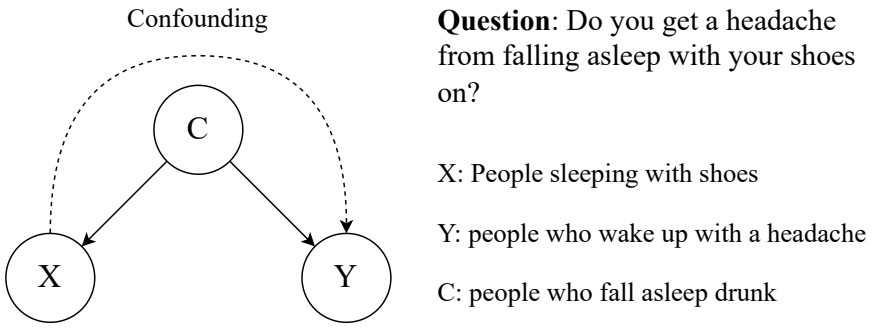


Figure 2.3: Example of confounding correlation between sleeping with shoes and getting a headache when you wake up.

To exemplify confounding variables and how do-calculus can avoid its associated complications, we use an example taken from Brady Neal's Youtube channel on Causal Inference [5]. Here we want to investigate if humans get a headache from falling asleep with their shoes on. If we investigate what percentage of people who sleep with shoes get headaches, we will find that it is a larger proportion than those who sleep without. This can lead to the misleading conclusion that sleeping with shoes can result in headaches.

If we investigate further, we will find that most people that sleep with their shoes on are drunk, and drunk people often wake up with a headache. Hence, although sleeping with shoes and waking up with a headache are correlated, there is no direct causal relationship between those two. Figure 2.3 makes this causal relationship more understandable. By conditioning on C, we will be able to avoid this being a confounding factor, but there will always be a possibility that other forgotten confounding variables exist. People on drugs or with neurological diseases fall under this category. Hence we need an approach that works regardless of how much we know about the system.

That is where the do-operator comes in. We start by randomly distributing the trial subjects into two groups. When the first group falls asleep, we put on their shoes and take them off for the second group. This makes the people who are drunk, on drugs, or have neurological diseases distribute evenly into the shoe- and shoeless sleepers category. Therefore, they will not confound the result, and we can confidently say that if the shoe sleepers had considerably higher instances of headache, then shoe sleeping leads to headaches. Let us say sleeping with shoes is equivalent to $X = 1$, sleeping without is $X = 0$, and waking up with a headache is equivalent to $Y = 1$, then we can express this mathematically as in Equation 2.12. Here $X \perp\!\!\!\perp Y$ means that X is independent of Y.

$$X \perp\!\!\!\perp Y \quad \text{if} \quad P(Y = 1 \mid do(X = 1)) = P(Y = 1 \mid do(X = 0)) \tag{2.12}$$

2.4 Global optimization solvers

Optimization problems are ubiquitous in many fields, ranging from engineering and physics to economics and finance [45]. Many of these problems involve finding a function's minimum or maximum value over a set of variables subject to constraints. A local optimization solver, such as gradient descent, can find a locally optimal solution in the vicinity of an initial guess. However, for many problems, there may be multiple local optima, and finding the global optimum becomes a more challenging task. In such cases, a global optimization solver is not recommended [45].

Global optimization solvers are specialized algorithms that attempt to find a function's global minimum or maximum over a given domain. Unlike local optimization algorithms, global optimization algorithms search the entire domain, often using intelligent strategies to identify promising regions for further exploration [45]. Global optimization algorithms are particularly useful in cases where the objective function is complex, nonlinear, or discontinuous.

One common approach to global optimization is Branch and Bound, a general algorithm that subdivides the domain into smaller subdomains and bounds the minimum or maximum value in each subdomain. By iteratively subdividing the domain and updating the bounds, Branch and Bound can eventually identify the global optimum within a desired tolerance [10]. Another approach is Simulated Annealing, which is inspired by the annealing process in metallurgy. Simulated Annealing start with a high temperature for the decision variable and gradually decreases it, allowing it to escape local minima and explore the entire domain [9]. Evolutionary algorithms, such as genetic algorithms and particle swarm optimization, use principles from evolutionary biology to search for global optima by iteratively evolving a population of potential solutions [28].

One distinguishing feature of global optimization methods is that they often require more computational resources than local optimization algorithms [18]. This is because the search space is much larger and more complex for global optimization problems, which often means that the algorithms must explore many more candidate solutions before finding the global optimum. Additionally, global optimization algorithms can require more sophisticated techniques for convergence testing and result validation to ensure that the global optimum has been found [20].

CHAPTER

Method 3

The Method chapter of this thesis will describe how CES explanation works and argue for the design choices of the Post-Hoc algorithm. The algorithm's approach can be divided into Time Frame Grouping, Backward Evaluation, and Forward Explanation. We discuss the purpose and functionality of these techniques in detail throughout the Method chapter. In simple terms, we employ Time Frame Grouping to group the number of decisions into a few. Then we use Backward Evaluation to see which states and actions had a significant effect on the outcome of the episode. Finally, we apply Forward Explanation to put it all together into a pleasing, humanly understandable explanation. There is a limited amount of prior research that was appropriate for addressing the issues presented in the thesis. All equations presented in the Method chapter are, therefore, derived by the author. For the ones that want to see an example of a CES explanation before reading the Method, you find it in Figure 4.4 in Chapter 4.

Before the implementation details, we will first discuss what makes an explanation good and investigate the causal structure of an agent acting on an environment.

3.1 What makes a good explanation

Before establishing what explanation we want from our algorithm, we need to discuss what makes an explanation good. This is both a philosophical and a scientific question. Due to the explanations' subjectiveness, we cannot define what lies in the word "good." [12]

One reason for this is that an explanation is situation-dependent, meaning that one explanation can be good or bad depending on the actual situation and the explanation receiver [12]. Explaining integrals to a high school student or someone studying at university is typically approached differently. A university-level explanation will likely go into much more detail, allowing for a deeper understanding.

Another challenge when formulating the explanation is to find the balance between accuracy and simplicity. Making an explanation simple and still 100 percent correct can often not be accomplished for complex problems. In real-world systems, causal relationships typically consist of millions of dependencies. As humans cannot comprehend such explanations, the explanation must be simplified, neglecting the parts of the influencing factors that have a low impact on the result.

To easier comprehend an explanation, humans can divide the system into subsystems [14]. Understanding parts of the system's behavior and finally putting the pieces together allows us to understand more complex patterns and contexts [14]. This idea is often referred to as Compositional Reasoning [14].

Finally, humans often gain understanding both through visual and textual explanations [37]. Having access to both types of explanation, therefore, generally speaking, strengthens our understanding.

3.2 Causal structure

The assumptions of how the causal structure of an episode is built up are critical to understanding CES explanations. Since the reward only has a purpose in the training process, it will not be a focus here. The same includes potential Q and value networks. With this out of the way, we are left with an actor taking actions based on its state and an environment producing the next state based on the action and state of the actor. Figure 3.1 illustrates what this might look like in a one-dimensional example.

The causal relationship becomes more intricate when multiple action and state dimensions are introduced, as depicted in Figure 3.2 for a single iteration with two-dimensional action and state spaces. Here the superscript represents the dimension index, and the subscript represents the iteration number. The number of edges per iteration has increased from 3 to 12, and as a result, an episode with hundreds of iterations would produce a causal graph with thousands of edges.

Furthermore, the actions and states can possess internal causal relationships in an individual iteration. E.g., if we want to make a left turn with a car, it only helps to turn the wheels to the left if the motor drives forward. The method will not explain these types of causal relationships.

The measures taken to simplify the graph are provided in Sections 3.3 and Section 3.4.

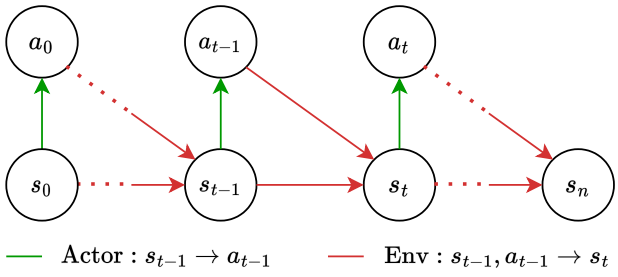


Figure 3.1: Action and state spaces causal relationship

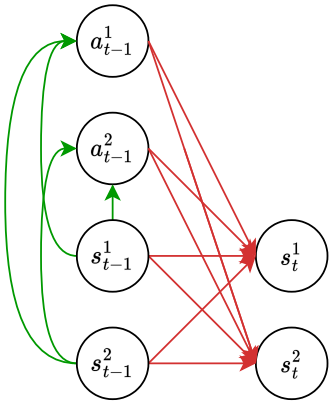


Figure 3.2: Two-dimensional action and state spaces causal relationship

3.3 Time Frame Grouping

One technique employed in CES explanations to reduce complexity is grouping actions into periods. This approach has the potential to simplify the problem without introducing significant errors. Particularly in physics-based environments, where the system's dynamics evolve continuously, short time intervals between actions typically result in minor state changes [26]. When the actor prediction function evolves continuously, similar inputs yield similar outputs. Therefore, for inputs (states) that are closely timed and similar, we can expect the corresponding actions to be similar as well. Note that CES explanations are not limited to systems that fall under this description, but the use of Time Frame Grouping, specifically, is mainly beneficial for such systems. To summarize, in physics-based environments with short time intervals between actions, there are often multiple subsequent actions that can be grouped and generalized into a single action.

Figure 3.3 shows what such a group might look like. Here the number of actions we needed to explain goes from 8 to 2. To achieve this, we use optimization techniques with some post-processing. Our problem is so distinct that we have no choice but to formulate our own equations from scratch to solve it.

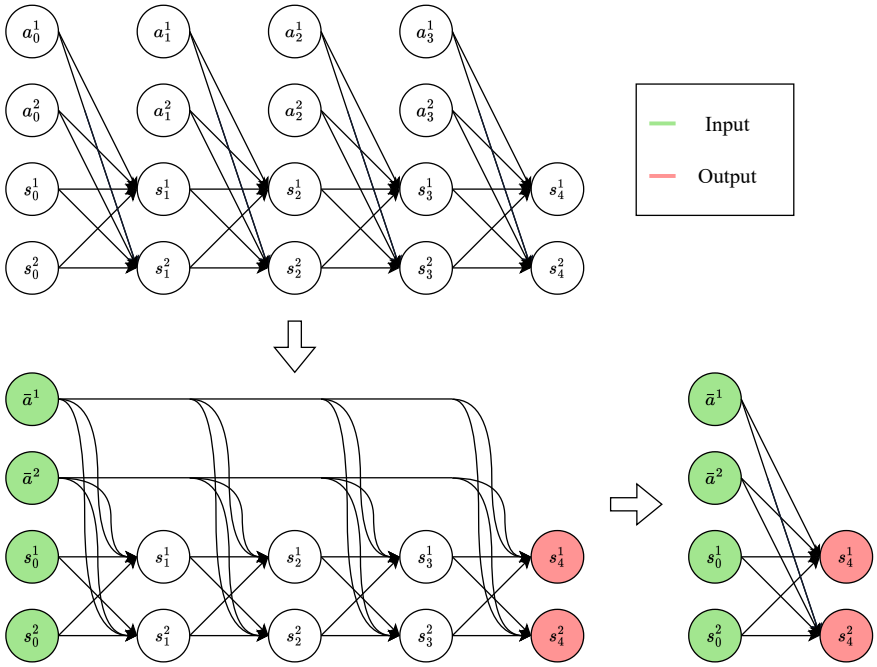


Figure 3.3: The image shows the reduction of complexity of grouping actions. $\bar{a}^1$ and $\bar{a}^2$ illustrates the average of a_0^1 to a_3^1 and a_0^2 to a_3^2 respectively.

3.3.1 Objective

First, we need to find an objective. Let us start with the case when we know how many splits (number of periods) we want for the episode actions, and the agent has a one-dimensional action space. Then Equation 3.1 would penalize the distance between the data measurements and the mean of the data to its corresponding time frame. Hence, we will get a low objective value by grouping similar subsequent data and a high one if we group data with high variance.

Figure 3.4 illustrates the idea behind each parameter more visually. We refer to the line between each period as split lines. T_i is the i'th period, σ_i is the variance of the data in the corresponding time frame, N is the index of the last period, n is the number of measurements, and X_j is the data measurement in iteration j. t_i is the sum of all periods before T_i, meaning that time frame i consist of all data between iteration t_i and $t_i + T_i$.

$$
\begin{aligned}
&\min_{T} \sum_{i=1}^{N-1} (T_i \sigma_i) + T_N \sigma_N \\
&\text{where:} \qquad T = [T_0, \cdots, T_i, \cdots T_{N-1}]^\top \\
&\qquad\qquad\qquad\qquad t_i = \sum_{k=1}^{i} T_k \\
&\qquad\qquad\qquad\qquad \sigma_i = \sum_{j=t_i}^{t_i+T_i} \frac{\left(X_j - \bar{X}_{T_i}\right)^2}{T_i} \\
&\qquad\qquad\qquad\qquad T_N = n - \sum_{i=1}^{N-1} T_i \\
&\text{constraint:} \qquad\qquad \sum_{i=1}^{N-1} T_i < n
\end{aligned}
\tag{3.1}
$$

If we make groups with actions close to each other, the variance is low and, thereby, also the

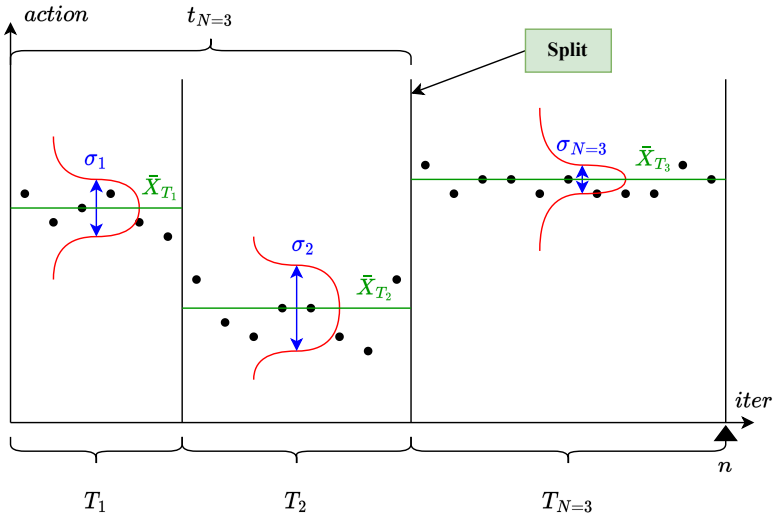


Figure 3.4: Illustrates all parameters in the 1D time-frame objective.

objective. To ensure equal weighting of each data point, we multiply with T_i: The greater the period, the higher the number of data points it encompasses. We guarantee that the splits are between the first and last iteration by constraining the sum of periods before T_N to be lower than the number of iterations.

Since the number of non-gradient optimization solvers that can handle inequality constraints is more limited, we modify the objective to be without constraints. Equation 3.2 is a version without constraints where we can guarantee that the global optimum has split lines inside the dataset. We can guarantee this because the variance stays the same when a time frame exceeds the last measurement, but the period increases. In this manner, we ensure that a solution exists within the measurements' bounds that equivalently groups the data but has a lower objective value.

Here, $c(T)$ is a function that significantly increases the loss when the constraint from Equation 3.1 is violated, assuming c_{const} is a high constant. The penalization helps to ensure that it does not converge to a optimum local outside of the measurements' bounds.

$$\min_T \sum_{i=1}^{N-1} (T_i \sigma_i) + T_N \sigma_N + c(T)$$

$$\text{where:} \qquad c(T) = \begin{cases} c_{const} \left| n - \sum_{i=1}^{N-1} T_i \right| & \text{if } \sum_{i=1}^{N-1} T_i > n \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Being able to know the ideal number of periods for a time series in advance is not something we can take for granted, and finding this should therefore happen automatically in the optimization function. Hence, to determine the ideal T's, it is necessary to assess various dimensions. The implementation in this thesis examines T's sizes between 1 and 8. Increasing T's dimensionality results in lower loss but greater complexity. To mitigate complexity penalization, we use Equation 3.3. This determines the total loss incurred by T and compares the minimum values across all permissible dimensions. c_D is a constant penalizing the increase of dimensions.

$$\min \left\{ \min_T \sum_{i=1}^{N-1} (T_i \sigma_i) + T_N \sigma_N + c(T) + c_D N \;\middle|\; N \in \{1, \cdots N_{max}\} \right\} \tag{3.3}$$

3.3.2 Solver

When we have designed the objective, the next stage is to find a fitting solver. For this purpose, we compare several solvers from the Scipy library. The first requirement for the solver is that it is a derivative-free solver. Nelder-Mead [30] and Particle Swarm [22] solvers are examples of this.

We use one-dimensional randomly generated step functions as data to test the different solvers. The total number of subsequent step functions in the data is seven, and each step varies in length and magnitude. Because we use step functions, we humans can easily verify if the time frame groups are logically positioned across the dataset. Specifically, each split should ideally be positioned in the transition between steps. Furthermore, we add some noise to the data to make the global solution less obvious for our solvers.

When testing such algorithms, we soon discover that the objective has multiple local optimums, which is way worse than the global optimum. Therefore, besides being derivative-free, it must also be a global solver. Since our objective is constraint-free, there are quite a few to choose from. Simplicial Homology Global Optimization (SHGO) [11], Basin-hopping [44], Differential Evolution [42], and Dual Annealing [46] are among these.

The criteria for our solver are to have a good running time and a low loss for different sizes of the T vector. Hence we experiment with different solvers to find the most well-suited. Figure 3.5 shows the result of testing the different solvers on the objective. Differential Evolution and Dual Annealing achieve low loss compared to the other methods, and Basin-Hopping is not guaranteed to find a solution. In the graph, we can see that Basin-Hopping could not find a solution with 3, 4, 5, 6, and 7 periods. According to the Scipy library, Basin-hopping aims to solve problems with a smooth objective function, which may be its problem [40]. Since reliability is essential for the use case, we will not consider Basin-Hopping further.

Regarding running time, SHGO and Differential Evolution are the fastest. Although Dual Annealing achieves low loss, the running time is relatively high compared to the others.

Based on the arguments above, we use Differential Evolution as our solver.

3.3.3 Post-processing

Even with the best solver, the result differs from what we desire. To get desirable results, we add two post-processing techniques. The first is dimensionality reduction to decrease the loss, and the second is frame merging to increase explanation simplicity.

Dimensionality reduction: With higher dimensionality, the solver tends to find better splits. This statement is based on a broad range of test experiments, and it can be challenging to build an intuition as to why. Nevertheless, we attempt to explain the subject matter in the best possible way. The more split lines that are spread across the dataset, the more likely it is that one of them is initialized close to a good split. A good split refers to a split that exhibits a significant difference in the average magnitude of the data on both sides. It will, therefore, considerably reduce the variance in the time frames on the left and right sides of the split. Although the statement is partly dependent on the solver: initial conditions closer to a good solution have a higher probability of converging toward it.

Our approach is, therefore, as follows: After getting the minimum values for all the dimensions, we start with the highest dimension and remove the worst split. Then we compare it with the solution from the lower dimension. If the solution is better than the previous one, we update the solution.

To make this easier to grasp, Figure 3.6 illustrates how the dimensionality reduction work. As shown, $N = 4$ has the best split, but $N = 2$ would give a lower loss if the solver had found the global solution.

We start with the highest dim ($N = 4$) and remove the worst split (the split after T_3). Next, we

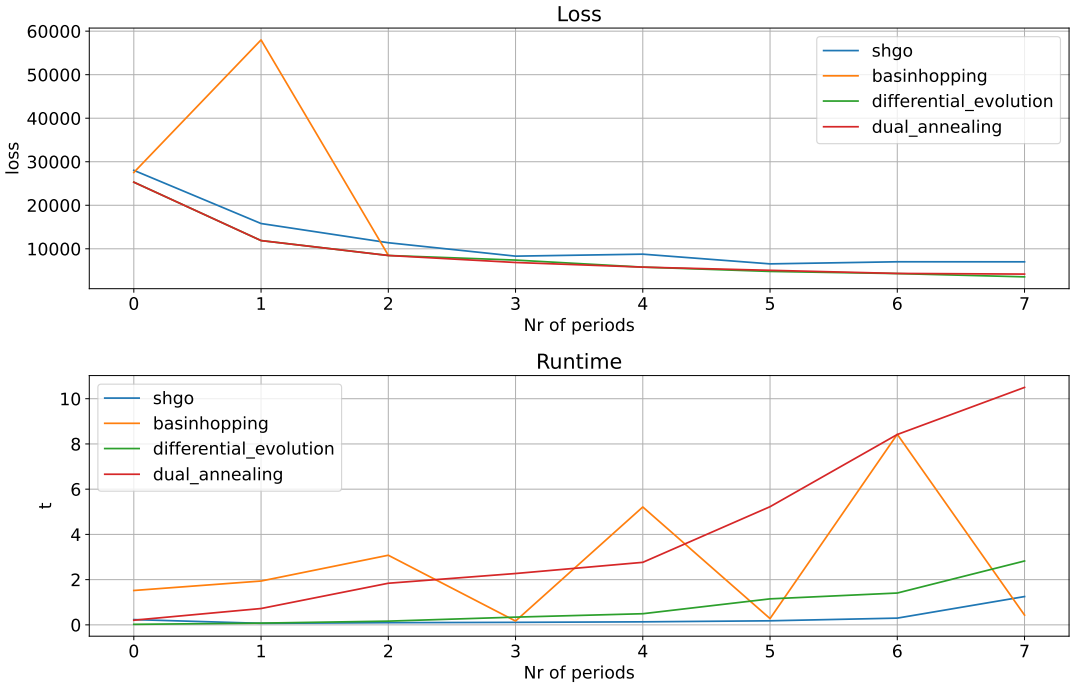


Figure 3.5: Comparing running time and loss for each optimizer where the x-axis shows the period dimentionality

compare this with $N = 3$. Since the loss is lower, we replace $N = 3$ with the new ($N_{new} = 3$). Then we remove another split line from $N = 4$ (the split after T_1) and compare it with the loss from $N = 2$. Since the loss is lower in the reduced form of $N = 4$, we also update $N = 2$. Although the Figure does not show it, the algorithm's next step would be to do the same process on $N = 3$ as on $N = 4$ and build itself down to $N = 1$. In the provided example, the optimal solution would have changed from $N = 3$ to $N = 2$.

Frame Merging: Since we need a new explanation for every unique splitting position, the number of explanations drastically increases when the action dimensionality is high. Figure 3.7 shows an example of groups created by the period selection from a_1 and a_2. As we can see, both G_2 and G_4 are very small and can be removed with minor period adjustments. The merging could be a part of the objective, but that would make it even more difficult for the solver to find a good solution. Therefore, we choose to include this in the post-processing.

The Frame Merging algorithm's procedure is given by the following. We move from iteration 0 to the last measurement and snap the closest split line from every action to the current iteration. It can be thought of as systematically scanning through every iteration to find good merges. Moving the split line will probably result in a higher loss, but the algorithm merges the split line if the loss is lower than a predefined limit. It will evaluate different subsets of the action space to see, e.g., if it is better to merge the first and second, the third and second, and so on. If it finds a merge and the new loss is lower than the last merge, it will update the split line to the new merging position.

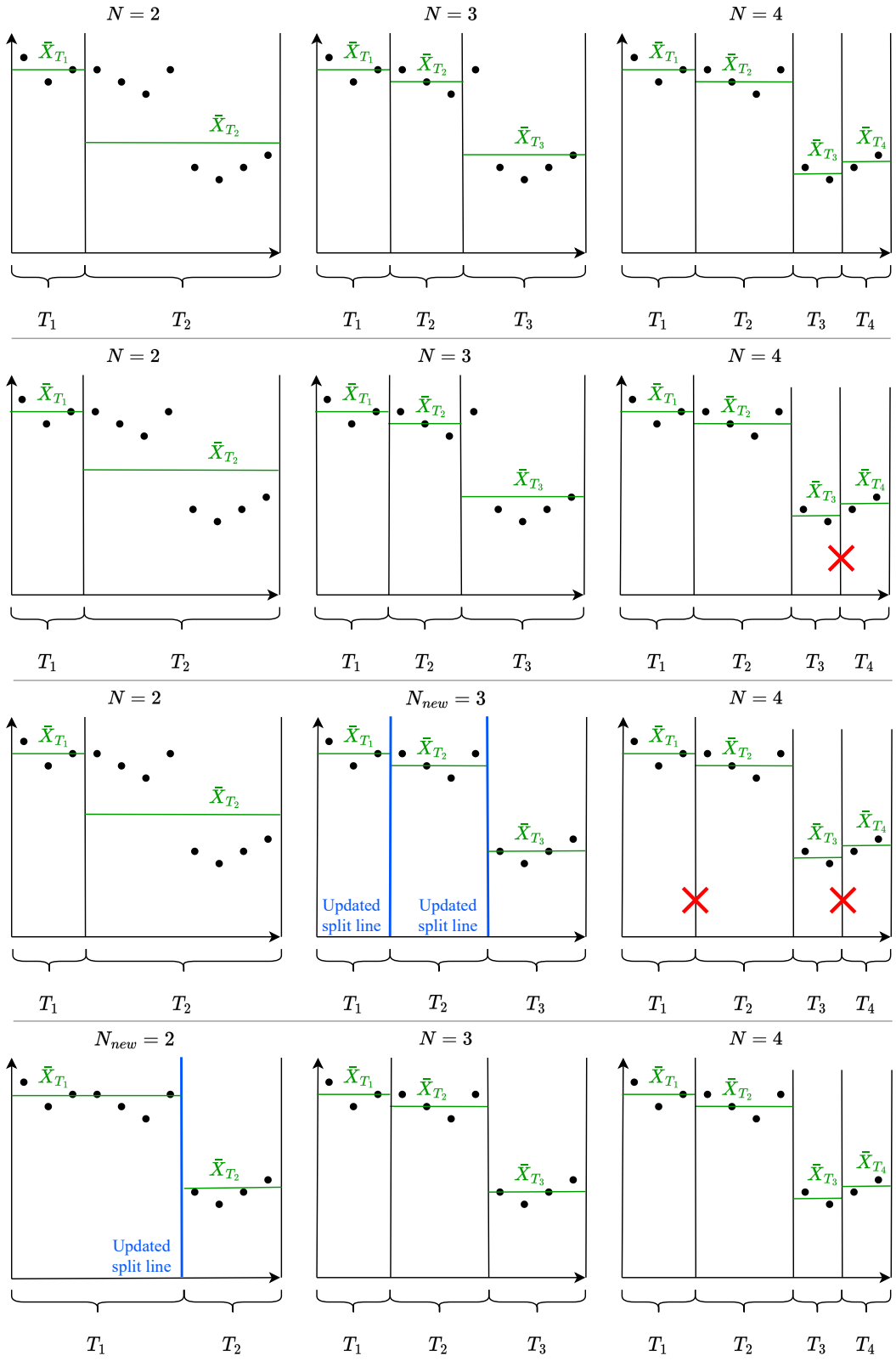


Figure 3.6: This is an example of a period reduction

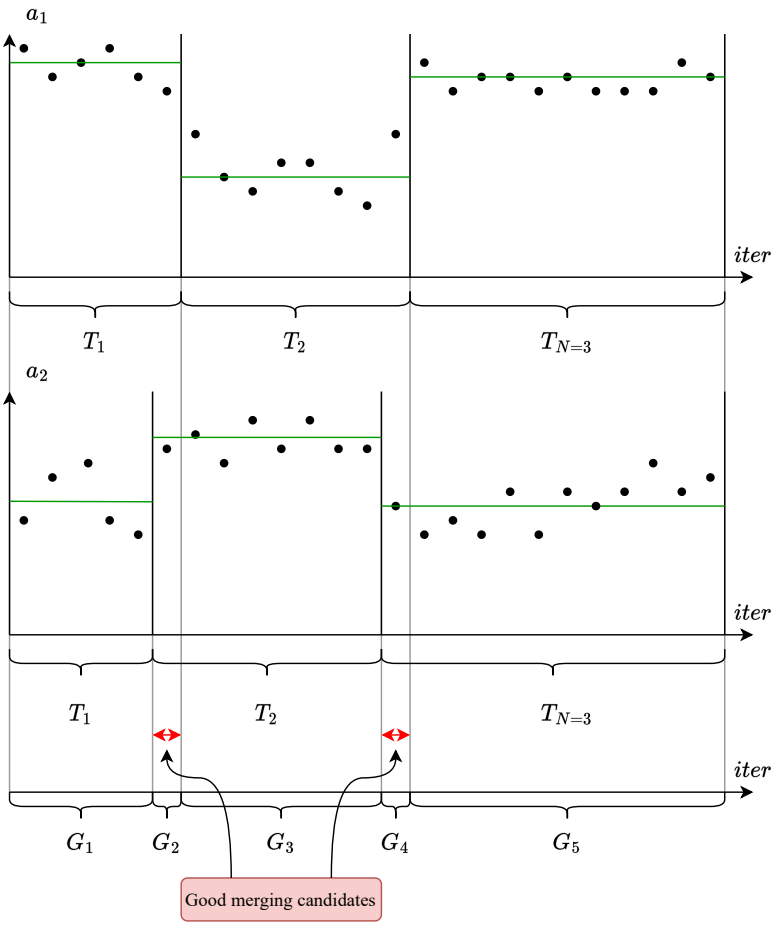


Figure 3.7: Here, we can see which groups the Frame Merging would remove.

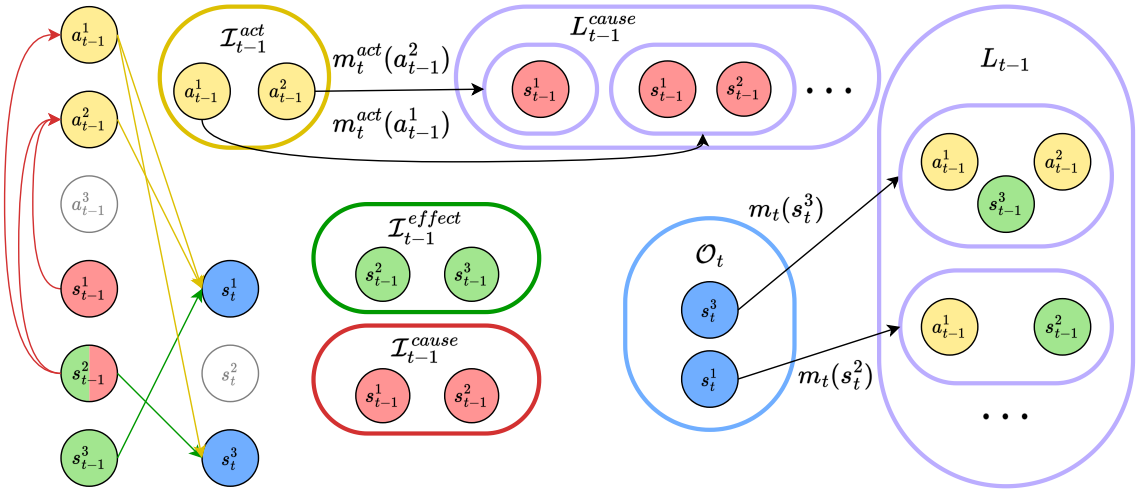


Figure 3.8: Terminology for backward evaluation. In the example, it is two active actions and three states for one time frame.

3.4 Backward Evaluation

When we have compressed the time frame into fewer ones, the next step is to reduce the number of edges for every time frame. We want to find the final state's causes by investigating which states and actions had the most influence. For every influential action, we find out the relevant states the agent considered (referred to as causes). Hence we start from the final state and build ourselves backward.

Be aware that t is used differently in Backward Evaluation and Forward Explanation than in Time Frame Grouping. Here t represents the iteration, meaning $s_{t=1}$ is the first measured state, $s_{t=2}$ is the second, etc. For ease of notation, we keep the time frame lengths equal to one to simplify the iteration-related subscripts in figures, equations, and pseudocode. This way, we inspect influential edges from, for example, $s_{t=1}$ to $s_{t=2}$ and not from $s_{t=1}$ to $s_{t=126}$, although the methodology is identical.

A desired property of the method is that the number of nodes does not systematically increase per iteration. If this were the case, the relevant nodes would eventually be so many that it would conceal the real reason for an outcome. We can prevent this by penalizing the number of nodes each time frame needs to explain. These nodes are referred to as $\mathcal{O}_t$ for iteration t.

Figure 3.8 helps to present some terminology created by the author and used throughout the report going forward. The figure illustrates one iteration where the algorithm has already removed unimportant edges. The grey nodes are nodes with no children or parents and are not a part of the explanation. The yellow nodes $\mathcal{I}^{act}_{t-1}$ are the set of actions with a high effect on $\mathcal{O}_t$. Each node in $\mathcal{I}^{act}_{t-1}$ has its causes, and they are colored red and referred to as $\mathcal{I}^{cause}_{t-1}$. The states colored in green with a high effect on $\mathcal{O}_t$ are referred to as $\mathcal{I}^{effect}_{t-1}$. $S^{\mathcal{S}}_t$ and $S^{\mathcal{A}}_t$ are sets of all states and actions, respectively, in iteration t (does also include the grey nodes). Equation 3.4 shows the relationship between $\mathcal{O}_t$, $\mathcal{I}^{effect}_t$ and $\mathcal{I}^{cause}_t$. Here n is the last iteration of the episode.

$$\mathcal{O}_t = \begin{cases} S^{\mathcal{S}}_t & \text{if } t = n \\ \mathcal{I}^{effect}_t \cup \mathcal{I}^{cause}_t & \text{otherwise} \end{cases} \tag{3.4}$$

We have two large sets containing subsets of nodes. These are L_{t-1} and L^{cause}_{t-1}. L_{t-1} contains all variations of children of a node in $\mathcal{O}_t$, while L^{cause}_{t-1} contains all variations of children of a node in $\mathcal{I}^{act}_{t-1}$. Consequently, we can sample the parents of $\mathcal{O}_t$ from L_{t-1} and the parents of $\mathcal{I}^{act}_{t-1}$ from L^{cause}_{t-1}.

To define the belonging sample of $\mathcal{O}_t$ and $\mathcal{I}_{t-1}^{act}$ from L_{t-1} and L_{t-1}^{cause}, we define the mapping functions m_t and m_t^{act}, respectively. Figure 3.8 shows the mapping of m_t and m_t^{act} on a sparse graph. Since m_t and m_t^{act} can express all combinations of edges for a given iteration, we treat these functions as decision variables in our optimization function that finds influential edges. Because of the unorthodox decision variables, we need an alternative way to solve the objectives. We present an objective used to find the influential edges on the final state in subsection 3.4.2 and 3.4.3, and the method of solving the objective in subsection 3.4.4.

3.4.1 Find value function for influential edges

Before going into the details of finding the influential edges for the environment steps and the actor model, we explain the core ideas for finding influential edges in general terms. Hence we use the term input instead of state and action and output instead of action and next state. In the upcoming subsections, we specify it to the actor model, with the state as input and action as output and the environment with states and actions as inputs and future states as outputs.

Our approach is to identify the input that leads to abnormal model behavior by leveraging a sample-based method. We send multiple samples to the model and observe the resulting outputs for different instances. To facilitate this, we maintain a buffer of recorded data from various runs of the actor interacting with the environment. Methods such as SHAP [25] and Causal SHAP [19] also use a similar approach, but these methods try to explain how important each input generally is, not how important a subset of the inputs is to the output.

To determine the abnormality of a given subset of input, we subtract its output from the average model output (the expected model behavior). We use do-calculus to identify which parts of the input are the most significant. Using $\mathbb{E}\left[f\left(\boldsymbol{X}\right) \mid do(\boldsymbol{X}_S = \boldsymbol{x}_S)\right] - \mathbb{E}\left[f\left(\boldsymbol{X}\right)\right]$, we determine how much a subset of the inputs makes the output shifts, either positively or negatively, from the average output. S is a subset of the inputs, and $\boldsymbol{X}$ is a random variable of the inputs. To identify the influential edges, we examine which subset of the input causes the most significant output shift in the same direction as the actual output. Doing so lets us pinpoint the crucial elements responsible for the model's abnormal behavior. Furthermore, we keep the edges from these highly influential inputs in the graph. Notice this preliminary value function is identical to the value function used in SHAP.

The value function is further used in the objective we want to maximize. Hence, we use the sign function to penalize shifting in the opposite direction of the actual output and reward a shift in the same direction. Altogether we get the value function in Equation 3.5 where $\boldsymbol{x}_N$ is all the inputs.

$$v(S) = \operatorname{sgn}\left(f\left(\boldsymbol{x}_N\right) - \mathbb{E}\left[f\left(\boldsymbol{X}\right)\right]\right)\left(\mathbb{E}\left[f\left(\boldsymbol{X}\right) \mid do(\boldsymbol{X}_S = \boldsymbol{x}_S)\right] - \mathbb{E}\left[f\left(\boldsymbol{X}\right)\right]\right) \tag{3.5}$$

3.4.2 Find environment edges

Value function

The value function for the environment is similar to the general case, and the formula can be found in Equation 3.6. Here f_o^{env} is constructed of step functions of the environment. The number of steps it takes equals the number of iterations in the current time frame. As a reminder, Figure 3.3 illustrates how the function consists of multiple steps compressed in one function. o is the output we want to explain where $o \in \mathcal{O}_t$. The inputs $\boldsymbol{x}$ consist of the states before the time frame and the mean action in the time frame.

$$v_o^{env}(S) = \operatorname{sgn}\left(f_o^{env}\left(\boldsymbol{x}_N\right) - \mathbb{E}\left[f_o^{env}\left(\boldsymbol{X}\right)\right]\right)\left(\mathbb{E}\left[f_o^{env}\left(\boldsymbol{X}\right) \mid do(\boldsymbol{X}_{S_o} = \boldsymbol{x}_{S_o})\right] - \mathbb{E}\left[f_o^{env}\left(\boldsymbol{X}\right)\right]\right) \tag{3.6}$$

Objective

For the objective, we want to maximize the value function and minimize the number of nodes in $\mathcal{O}_t$. Since we cannot access $\mathcal{O}_t$ before the actor edges are available, we need to base the objective on $\mathcal{I}_{t-1}^{act}$ and $\mathcal{I}_{t-1}^{effect}$ instead. $\mathcal{I}_{t-1}^{effect}$ has a direct correlation with $\mathcal{O}_{t-1}$'s size while $\mathcal{I}_{t-1}^{act}$ has a more indirect correlation ($\mathcal{I}_{t-1}^{act}$ is correlated to $\mathcal{I}_{t-1}^{cause}$, which is correlated to $\mathcal{O}_{t-1}$). Therefore we want a function that penalizes the size of $\mathcal{I}_{t-1}^{act}$ and $\mathcal{I}_{t-1}^{effect}$. This function is called g and is in Equation 3.8. c_{act}, c_{effect}, and c_p^g are positive constants weighing the penalty importance of additional action and state nodes.

The objective sums over the value function for all outputs $\mathcal{O}_t$ and multiplies it with g. Altogether this gives Equation 3.7. The summation of value functions maximizes the explanation accuracy, while g maximizes the simplicity.

$$\max_{m_t} \kappa_t^{env}(\mathcal{O}_t) = \max_{m_t} g\left(\left|\mathcal{I}_{t-1}^{act}\right|, \left|\mathcal{I}_{t-1}^{effect}\right|; c_{act}, c_{effect}, c_p^g\right) \sum_{o \in \mathcal{O}_t} \upsilon_o^{env}\left(m_t(o)\right)$$

where:

$$\begin{aligned} g' < 0, g > 0, \\ m_t : \mathcal{O}_t \to L_{t-1} \\ \mathcal{I}_{t-1}^{act} = \textstyle\bigcup_{o \in \mathcal{O}_t} m(o) \cap S_{t-1}^{\mathcal{A}} \\ \mathcal{I}_{t-1}^{effect} = \textstyle\bigcup_{o \in \mathcal{O}_t} m(o) \cap S_{t-1}^{\mathcal{S}} \\ \mathcal{O}_t \subseteq S_t^{\mathcal{S}} \\ L_{t-1} = \left\{ S \mid S \subseteq S_{t-1}^{\mathcal{S}} \cup S_{t-1}^{\mathcal{A}} \right\} \end{aligned} \tag{3.7}$$

$$g\left(\left|\mathcal{I}_t^{act}\right|, \left|\mathcal{I}_t^{effect}\right|; c_{act}, c_{effect}, c_p^g\right) = \frac{1}{\left(c_{act}\left|\mathcal{I}_t^{act}\right| + c_{effect}\left|\mathcal{I}_t^{effect}\right|\right)^{c_p^g}} \tag{3.8}$$

3.4.3 Find actor edges

Value function

The value function of the actor has the same structure as the environment. Here f_i^{act} is the actor function (probably a neural network) where i is the action in $\mathcal{I}_{t-1}^{act}$, which we want to explain. Since we use the mean action as environment input for each time frame, we want to explain the state in the time frame that gives an actor output as close to the mean action as possible. Altogether we get Equation 3.9.

$$\upsilon_i^{act}(S) = \operatorname{sgn}\left(f_i^{act}(\boldsymbol{x}_N) - \mathbb{E}\left[f_i^{act}(\boldsymbol{X})\right]\right)\left(\mathbb{E}\left[f_i^{act}(\boldsymbol{X}) \mid do(\boldsymbol{X}_{S_i} = \boldsymbol{x}_{S_i})\right] - \mathbb{E}\left[f_i^{act}(\boldsymbol{X})\right]\right) \tag{3.9}$$

Objective

The objective of the actor has a similar structure as the environment. Here we want to penalize the additional nodes to $\mathcal{O}_t$ from $\mathcal{I}_t^{cause}$. These nodes are equivalent to $\mathcal{I}_t^{cause} \setminus \mathcal{I}_t^{effect}$. The penalization function h of $\mathcal{I}_t^{cause}$ is found in Equation 3.11. c_{cause} and c_p^h serves the same purpose as c_{act}, c_{effect} and c_p^g; to penalize extra nodes in the explanations. We take the sum of the value function for all actions, similar to the environment objective. Altogether we get the objective in Equation 3.10.

$$\max_{m_t^{act}} \kappa_t^{act}(m_t^{act}) = \max_{m_t^{act}} h(|\mathcal{I}_t^{cause} \setminus \mathcal{I}_t^{effect}|; c_{cause}, c_p^h) \sum_{i \in \mathcal{I}_t^{act}} v_i^{act}\left(m_t^{act}(i)\right)$$

where:

$$\begin{aligned} h' < 0, h > 0, \\ m_t^{act} : \mathcal{I}_t^{act} \to L_t^{cause} \\ \mathcal{I}_t^{cause} = \textstyle\bigcup_{i \in \mathcal{I}_t^{act}} m_t^{act}(i) \\ L_t^{cause} = \left\{ S \;\middle|\; S \subseteq S_t^{\mathcal{S}} \right\} \end{aligned} \tag{3.10}$$

$$h(|\mathcal{I}_t^{cause} \setminus \mathcal{I}_t^{effect}|; c_{cause}, c_p^h) = \frac{1}{\left(c_{cause}|\mathcal{I}_t^{cause} \setminus \mathcal{I}_t^{effect}|\right)^{c_p^h}} \tag{3.11}$$

3.4.4 Probabilistic objective sampling

Finding a solver for κ_t is more complex than it was finding a solver for the time frames. Since we have a decision function m_t and not a decision variable, we can not place it in a coordinate system. This prevents us from using Nelder–Mead, Quasi-Newton, or any other general method except for brute force. For brute force, we need to test $2^{|S_{t-1}^{\mathcal{S}} \cup S_{t-1}^{\mathcal{A}}|}$ combinations of influential nodes to each combinations of $\mathcal{O}_t$. The number of combinations we can have for our objective is, therefore, $2^{|S_{t-1}^{\mathcal{S}} \cup S_{t-1}^{\mathcal{A}}| \cdot |\mathcal{O}_t|}$, and we can not test every combination. Instead, we need to make a solver to our specific objective. We do this by sampling m_t mappings that probably give a high κ_t value and keep the highest κ_t. Note, we do the exact same procedure to sample m_t^{act} as for m_t, and we will, therefore, only phrase the reasoning based on m_t. Despite that, we give the relevant formulas to sample m_t^{act} and provide the pseudocode for both in the upcoming subsection.

As explained earlier, m_t maps input node $o \in \mathcal{O}_t$ to a set $S_o \in L_{t-1}$. Therefore, finding a preliminary κ_t estimate based on *one* input o is a good start. Equation 3.12 estimates this very concept. Here we assume that the return value for the value function with S_o as input would be the average return value for all $o \in \mathcal{O}_t$. This makes $\sum_{o \in \mathcal{O}_t} v_o^{env}\left(m_t(o)\right) = |\mathcal{O}_t| \cdot v_o^{env}\left(S_o\right)$.

$$\tilde{\kappa}_i^{env}(S_o) = \tilde{g}(S_i; c_{act}, c_{effect}) \cdot |\mathcal{O}| \cdot v_o^{env}\left(S_o\right) \tag{3.12a}$$

$$\tilde{\kappa}_i^{act}(S_i^{act}) = \tilde{h}(S_i; c_{cause}) \cdot |\mathcal{I}_{act}| \cdot v_i^{act}\left(S_i^{act}\right) \tag{3.12b}$$

The next step is to find reasonable estimates of g and h. We start with g since it is the easiest. The function of $\tilde{g}$ is the same as g but with estimates of the sizes $\left|\mathcal{I}_{t-1}^{act}\right|$ and $\left|\mathcal{I}_{t-1}^{effect}\right|$. As mentioned earlier, $\mathcal{I}_{t-1}^{act}$ is the nodes returned from m_t that are actions, and $\mathcal{I}_{t-1}^{effect}$ is the nodes returned from m_t that are states. To find an estimate of $\left|\mathcal{I}_{t-1}^{effect}\right|$, we can multiply the probability of an arbitrary state being one of m_t's possible return values by the number of states. The probability of an arbitrary state being one of m_t's possible return values is equivalent to 1 minus the probability of it not being a part of it ($i \notin \mathcal{I}_{t-1}^{effect}$). This probability is equal to $\left(1 - \left(1 - \left|S_o \cap S^{\mathcal{S}}\right| / \left|S^{\mathcal{S}}\right|\right)^{|\mathcal{O}|}\right)$ when assuming $|S_i|$ is the expected return size of m_t. For a detailed explanation, see Appendix, Section A.1. The same logic and formula apply to the actions only by changing $S^{\mathcal{S}}$ to $S^{\mathcal{A}}$. Altogether this gives Equation 3.13.

$$\tilde{g}(S_o; c_{act}, c_{effect}, c_p^g) = g\left(\tilde{\mathcal{I}}_{act}^{size}, \tilde{\mathcal{I}}_{effect}^{size}; c_{act}, c_{effect}, c_p^g\right) \tag{3.13a}$$

$$\tilde{\mathcal{I}}_{act}^{size} = \left(1 - \left(1 - \frac{\left|S_o \cap S^{\mathcal{A}}\right|}{|S^{\mathcal{A}}|}\right)^{|\mathcal{O}|}\right) \left|S^{\mathcal{A}}\right| \tag{3.13b}$$

$$\tilde{\mathcal{I}}^{size}_{effect} = \left(1 - \left(1 - \frac{\left|S_o \cap S^{\mathcal{S}}\right|}{|S^{\mathcal{S}}|}\right)^{|\mathcal{O}|}\right)\left|S^{\mathcal{S}}\right| \tag{3.13c}$$

To find a reasonable estimate of h, we need to find the expected number of added states in $\mathcal{O}_{t-1}$ due to action causes. We refer to the expected return size of m_t^{act} as $|S_i^{act}|$. As earlier, we find this by multiplying the likelihood that a cause adds an arbitrary state to $\mathcal{O}_{t-1}$ with the number of states. For the cause to add a state, it cannot already exist in $\mathcal{O}_{t-1}$. $1 - |\mathcal{I}_{t-1}^{effect}|/|S^{\mathcal{S}}|$ is the likelihood that it does not exist in $\mathcal{I}_{t-1}^{effect}$ and therefore also in $\mathcal{O}_{t-1}$. The likelihood that this node is a part of any cause is $1-\left(1 - |S_i^{act}| \,/\, \left|S^{\mathcal{S}}\right|\right)^{|\mathcal{I}_t^{act}|}$ given that $S_i^{act} \in L_t^{cause}$ has the expected size of all m_t^{act}'s return values. The reasoning for this is the same as for $\tilde{\mathcal{I}}^{size}_{effect}$ only that the input nodes of m_t^{act} is $\mathcal{I}_t^{act}$ instead of $\mathcal{O}_t$, and the estimated return size of the sets for m_t^{act} is $|S_i^{act}|$ instead of $\left|S_o \cap S^{\mathcal{S}}\right|$. Finally, the likelihood that an arbitrary node is an action cause and not already in $\mathcal{I}_{t-1}^{effect}$ is $\left(1 - |\mathcal{I}_t^{effect}|/|S^{\mathcal{S}}|\right)\left(1 - \left(1 - |S_i^{act}| \,/\, \left|S^{\mathcal{S}}\right|\right)^{|\mathcal{I}_t^{act}|}\right)$. Altogether this gives Equation 3.14.

$$\tilde{h}(S_i; c_{cause}, c_p^h) = h(\tilde{\mathcal{I}}^{size}_{new\ cause}; c_{cause}, c_p^h) \tag{3.14a}$$

$$\tilde{\mathcal{I}}^{size}_{new\ cause} = \left(1 - \frac{|\mathcal{I}_t^{effect}|}{|S^{\mathcal{S}}|}\right)\left(1 - \left(1 - \frac{|S_i^{act}|}{|S^{\mathcal{S}}|}\right)^{|\mathcal{I}_t^{act}|}\right)\left|S^{\mathcal{S}}\right| \tag{3.14b}$$

Next, we want to find combinations of $S_o \in L_{t-1}$ for all $o \in \mathcal{O}_t$. The higher the κ_t estimate S_o has, the likelier it should be a return value of mapping function m_t. We achieve this by making a uniform probability density function to sample outputs $m_t(o) = S_o$. Each S_o with a positive κ_t estimate has an associated range in the uniform distribution. See Figure 3.9. The $\tilde{\kappa}_t$ value is proportional to its associated length. This way, it is likely to sample in the range of a high-valued kappa estimate.

When we sample, and the value ends up at $S_{1,2}$'s and $S_{2,3}$'s range as in the figure, we calculate the loss for $m_t(o_1) = S_{1,2}$ and $m_t(o_2) = S_{2,3}$. We keep sampling for a fixed number of iterations and store the one with the highest objective value.

We use the same technique of finding the influential environment edges for the actor.

3.4.5 Implementation

As for the implementation of Backward Evaluation, it has been decided to provide three blocks of pseudocode to make the algorithm easier to understand. The first shows the overall method and details how the causal graph is constructed based on m_t, and the second and third show how we find m_t^{act} and m_t using probabilistic objective sampling. We find the first, second, and third pseudocode in Algorithm 1, Algorithm 2, and Algorithm 3, respectively.

As illustrated in Algorithm 1, we start by initializing a graph only consisting of the final nodes. Then, we iterate backward from the final state. We initialize new state and action nodes for every iteration and decide whom to keep based on the found m_t and m_t^{act}. The max number of iterations when finding m_t and m_t^{act} is decided in the initialization of the algorithm. Since finding κ_t estimates is generally the most time-consuming operation in the CES explanation, we can set the max iteration high without affecting the total running time that much. In the thesis's implementation, it was set to 100000. We divide the edges into three: Pointing from a state to the next state $E^{\mathcal{S}\to\mathcal{S}}$, pointing from an action to the next state $E^{\mathcal{A}\to\mathcal{S}}$, and pointing from a state to an action $E^{\mathcal{S}\to\mathcal{A}}$. Each gets added to the graph for every iteration.

For the actual implementation, we represent the graph as a tree structure. Generally, graphs can be implemented and presented in numerous ways, such as adjacency lists and adjacency matrices [43]. Implementing it as a tree has several advantages for our causal graph. E.g., each node is implemented as a class. Therefore we can store several values of different types in each node. In

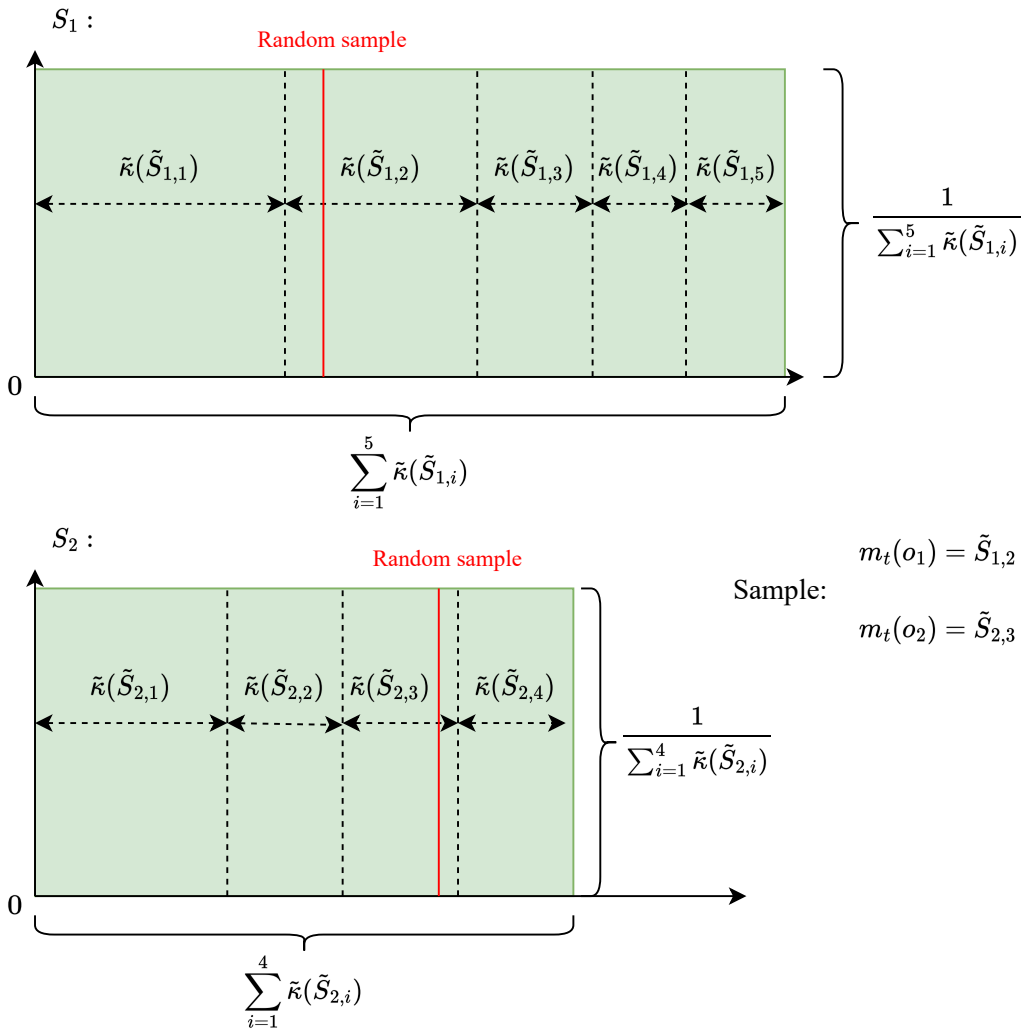


Figure 3.9: This figure shows how we sample subsets S from a uniform distribution when $|\mathcal{O}_t| = 2$ based on an estimate of the objective. The uniform distribution is represented as a green area. As we can see, the higher estimate of the objective, the likelier it is to be sampled.

our implementation, we store if the node is an action or a state, which iteration the node belongs to, a unique ID for the state or action, and its parents and children. Compared to an adjacency matrix, we spear plenty of memory. An adjacency matrix stores all possible edges in a matrix [43]. With no modifications to the method, it will take memory for potential edges between all iterations, although the causal structure does not allow that. The size of the matrix will therefore be huge. A pro of using an adjacency matrix is that we can check if an edge exists in constant time. An adjacency list uses similarly much space as a three. When that is said, it does not have the causal hierarchy between the nodes as easily accessible.

Algorithm 1 Get Simplified Graph G of environment causal effects

Initialize Network parameters $G = (V = \{S_n^{\mathcal{S}}\},\ E = \emptyset),\ \mathcal{O}_t = S_n^{\mathcal{S}},$ MaxIter, n_states, n_actions
for each t in reverse episode order **do**
  $S_{t-1}^{\mathcal{S}} \leftarrow$ CreateNewNodes(Size = n_states, time $= t-1$, is_state = True)
  $S_{t-1}^{\mathcal{A}} \leftarrow$ CreateNewNodes(Size = n_states, time $= t-1$, is_state = False)
  $m_t \leftarrow$ FindM($\mathcal{O}_t,\ S_{t-1}^{\mathcal{S}},\ S_{t-1}^{\mathcal{A}},$ MaxIter)
  $\mathcal{I}_{t-1}^{effect} = \bigcup_{o \in \mathcal{O}_t} m_t(o) \cap S^{\mathcal{S}}$
  $\mathcal{I}_{t-1}^{act} = \bigcup_{o \in \mathcal{O}_t} m_t(o) \cap S^{\mathcal{A}}$
  $m_{t-1}^{act} \leftarrow$ FindMAct($\mathcal{I}_{t-1}^{act},\ S_{t-1}^{\mathcal{S}},\ S_{t-1}^{\mathcal{A}},$ MaxIter)
  $\mathcal{I}_{t-1}^{cause} \leftarrow \bigcup_{i=1}^{|M_{act}|} S_i^{act}$
  $E^{\mathcal{A}\to\mathcal{S}} \leftarrow \left\{ (x,y) \;\middle|\; x \in \mathcal{I}_{t-1}^{act}, y \in \mathcal{O}_t \right\}$
  $E^{\mathcal{S}\to\mathcal{S}} \leftarrow \left\{ (x,y) \;\middle|\; x \in \mathcal{I}_{t-1}^{effect}, y \in \mathcal{O}_t \right\}$
  $E^{\mathcal{S}\to\mathcal{A}} \leftarrow \left\{ (x,y) \;\middle|\; x \in \mathcal{I}_{t-1}^{cause}, y \in \mathcal{I}_{t-1}^{act} \right\}$
  $G.V \leftarrow G.V \cup \mathcal{I}_{t-1}^{act} \cup \mathcal{I}_{t-1}^{effect} \cup \mathcal{I}_{t-1}^{cause}$
  $G_S.E \leftarrow G_S.E \cup E^{\mathcal{A}\to\mathcal{S}} \cup E^{\mathcal{S}\to\mathcal{S}} \cup E^{\mathcal{S}\to\mathcal{A}}$
  $\mathcal{O}_{t-1} \leftarrow \mathcal{I}_{t-1}^{effect} \cup \mathcal{I}_{t-1}^{cause}$
end for

Algorithm 2 and 3 are so similar that we will explain these together. These algorithms show the procedure of finding m_t and m_t^{act}. As shown, its starts by finding κ_t estimates. We used multiprocessing to fill the κ_t estimate values in the actual code. Hence, divide the κ_t estimate array into smaller arrays, where each core is responsible for filling in its array with κ_t estimates. The Algorithm will merge these arrays when these processes finish. Doing this increased the speed by approximately three times with an eight-core CPU and the lunar lander environment for the author's implementation. κ_t^{act} only uses a single core since it is not a significant time to save (the environment step function is more computer heavy than a forward pass on the actor's network).

When the algorithm finds the κ_t estimates, it begins the probabilistic objective sampling. The sampling keeps searching for the global solution until it has reached its maximum iteration. All samples will be passed through the objective function to see how good the estimate actually is. The algorithm will reuse the value function returns found in the κ_t estimate to save computing time. Finally, it will return its best-found solution.

Algorithm 2 This is a function gives a good estimate of the optimal m_t

function FINDM($\mathcal{O}_t,\ S^{\mathcal{S}}_{t-1},\ S^{\mathcal{A}}_{t-1},$ MaxIter)
  $\tilde{\kappa}_t \leftarrow [\emptyset]_{|\mathcal{O}_t| \times |L_{t-1}|}$ ▷ A 2D list of null pointers
  $o_{idx} \leftarrow 1$
  for $o \in \mathcal{O}_t$ **do**
    $l_{idx} \leftarrow 1$
    for $S \in L_{t-1}$ **do**
      $\tilde{\kappa}_t[o_{idx}, l_{idx}] \leftarrow \left(S,\ \text{objEst} = \tilde{g}(S; c_{act}, c_{effect}, c^g_p) \cdot \upsilon_o(S),\ \text{ValFunc} = \upsilon_o(S)\right)$
      $l_{idx} \leftarrow l_{idx} + 1$
    end for
    $o_{idx} \leftarrow o_{idx} + 1$
  end for
  Distributions $\leftarrow$ CONVERTESTTODISTRIBUTION($\tilde{\kappa}_t$)
  $m_t \leftarrow$ Distributions.getMWithMaxKappaEst() ▷ The initial value is the most likely solution
  objMax $\leftarrow$ ENVOBJECTIVE($m_t, \mathcal{O}_t$)
  for MaxIter **do**
    $\tilde{m}_t \leftarrow$ Distributions.Sample()
    objS $\leftarrow$ ENVOBJECTIVE($m_t, \mathcal{O}_t$)
    if objS $\geq$ objMax **then**
      $m_t \leftarrow \tilde{m}_t$
      objMax $\leftarrow$ objS
    end if
  end for
  return m_t
end function

Algorithm 3 This is a function gives a good estimate of the optimal m^{act}_t

function FINDMACT($\mathcal{O}_t,\ S^{\mathcal{S}}_{t-1},\ S^{\mathcal{A}}_{t-1},$ MaxIter)
  $\tilde{\kappa}_t \leftarrow [\emptyset]_{|\mathcal{O}_t| \times |L^{cause}_{t-1}|}$ ▷ A 2D list of null pointers
  $o_{idx} \leftarrow 1$
  for $o \in \mathcal{O}_t$ **do**
    $l_{idx} \leftarrow 1$
    for $S \in L^{cause}_{t-1}$ **do**
      $\tilde{\kappa}_t[o_{idx}, l_{idx}] \leftarrow \left(S,\ \text{objEst} = \tilde{h}(S; c_{cause}, c^h_p) \cdot \upsilon_i(S),\ \text{ValFunc} = \upsilon_i(S)\right)$
      $l_{idx} \leftarrow l_{idx} + 1$
    end for
    $o_{idx} \leftarrow o_{idx} + 1$
  end for
  Distributions $\leftarrow$ CONVERTESTTODISTRIBUTION($\tilde{\kappa}_t$)
  $m^{act}_t \leftarrow$ Distributions.getMWithMaxKappaEst() ▷ The initial value is the most likely solution
  objMax $\leftarrow$ ACTOBJECTIVE($m_t, \mathcal{O}_t$)
  for MaxIter **do**
    $\tilde{m}^{act}_t \leftarrow$ Distributions.Sample()
    objS $\leftarrow$ ACTOBJECTIVE($m_t, \mathcal{O}_t$)
    if objS $\geq$ objMax **then**
      $m^{act}_t \leftarrow \tilde{m}^{act}_t$
      objMax $\leftarrow$ objS
    end if
  end for
  return m^{act}_t
end function

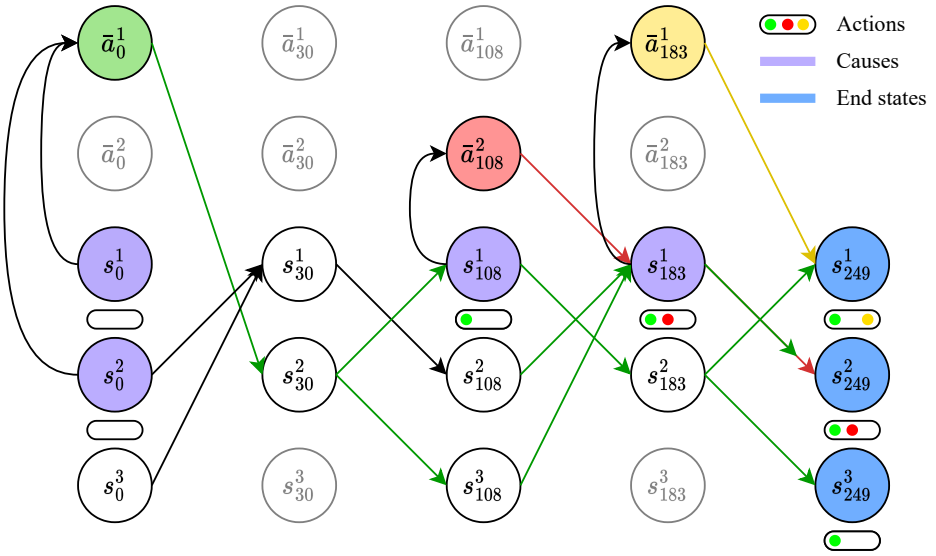


Figure 3.10: This figure helps to illustrate how to go from a graph to a textual explanation using the Forward Explanation procedure

3.5 Forward Explanations

When we have a sparse explanation graph, the next step is to produce a human-like explanation from it. Contrary to the Backward Evaluation, we do this in a forward manner, where we start from the initial condition and end at the final state. We aim to get a textual explanation based on the graph structure with an image of the environment for every explained state. Thereby, the explanation would look more or less like a cartoon strip.

Essential qualities of the textual explanation are that the explanation is divided into logical sub-explanations and that the sub-explanations are coherent. To find the best-suited sub-explanations, we must evaluate what parts of the episodes we want to interpret. We narrow our interests down to the actions made by the actor to achieve the end state and the state the actor considered when it took its actions.

We refer to the states that the actor considers as causes and the impacts that the actor's actions have on the environment as the effects. To make the explanation coherent, we need to relate every cause and final state to its origin. The origin is either caused by one or more effects of the previous actions or the initial condition.

To understand the implementation of the algorithm and how to produce a textual result from it, we use the example provided in Figure 3.10. As we can see, the node that are causes are colored in purple, the end states are colored in blue, and the first, second, and third action is colored in green, red, and yellow, respectively. Under the causes and end states is a tag containing which action it resulted from. If a node has a tag with green and red colors, it results from the first and second actions, if it has green and yellow, it is a result of the first and third, and so on. To present the effects of an action clearer, we color the edges pointing from an action with the same color as the action node itself. We can see from the subscript that the time frames here are from iteration 0 to 30, 30 to 108, 108 to 183, and 183 to 249.

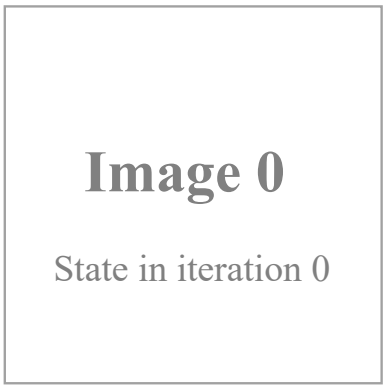
Now let us go into the implementation details. The algorithm starts at the initial state, which would be the root if the graph were implemented as a tree structure. The algorithm will move forward to the next layer of nodes until it finds its first influential action to explain. We can get the action causes by retrieving its parents. To find its effects, we traverse down the tree and only store the nodes that are either a cause or an end state. The algorithm does these steps until we reach the final state.

The algorithm will represent the cause and effect as in Figure 3.11. Since the value function

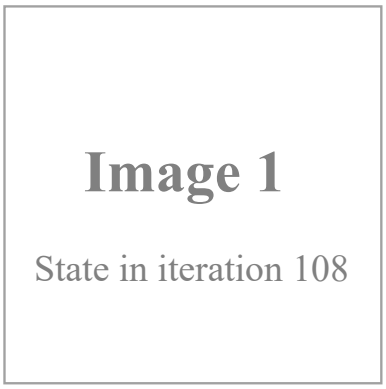
compare the node to its expected value, we illustrate the expected value of the nodes at the top of the explanation. This way, we can do the same. E.g., $\bar{a}_0^1$ is higher than its expected value because $s_0^1 = \square$ and $s_0^2 = \square$. We uniquely colorize the time frame, state, cause, and effect subtitle to increase the explanation's interpretability. For an actual environment with an agent, we would include images of the agent at the first state of its time frame and define the values for the states and actions.

Episode explanation:

Expected state values: $\bar{s}^1 = \square$, $\bar{s}^2 = \square$, $\bar{s}^3 = \square$
Expected action values: $\bar{a}^1 = \square$, $\bar{a}^2 = \square$



Time frame: 0-30
State: $s_0^1 = \square$, $s_0^2 = \square$, $s_0^3 = \square$
Cause: Do $\bar{a}_0^1 = \square$ because of $s_0^1 = \square$, $s_0^2 = \square$
Effect: $\bar{a}_0^1 = \square$ contributes to $s_{108}^1 = \square$, $s_{183}^1 = \square$, $s_{249}^1 = \square$, $s_{249}^2 = \square$, $s_{249}^3 = \square$



Time frame: 108-183
State: $s_{108}^1 = \square$, $s_{108}^2 = \square$, $s_{108}^3 = \square$
Cause: Do $\bar{a}_{108}^2 = \square$ because of $s_{108}^1 = \square$
Effect: $\bar{a}_{108}^2 = \square$ contributes to $s_{183}^1 = \square$, $s_{249}^2 = \square$

Time frame: 183-249
State: $s_{183}^1 = \square$, $s_{183}^2 = \square$, $s_{183}^3 = \square$
Cause: Do $\bar{a}_{183}^1 = \square$ because of $s_{183}^1 = \square$
Effect: $\bar{a}_{183}^1 = \square$ contributes to $s_{249}^1 = \square$

Iteration: 249
State: $s_{249}^1 = \square$, $s_{249}^2 = \square$, $s_{249}^3 = \square$

Figure 3.11: The explanation of graph in Figure 3.10 using the Forward Explanation algorithm

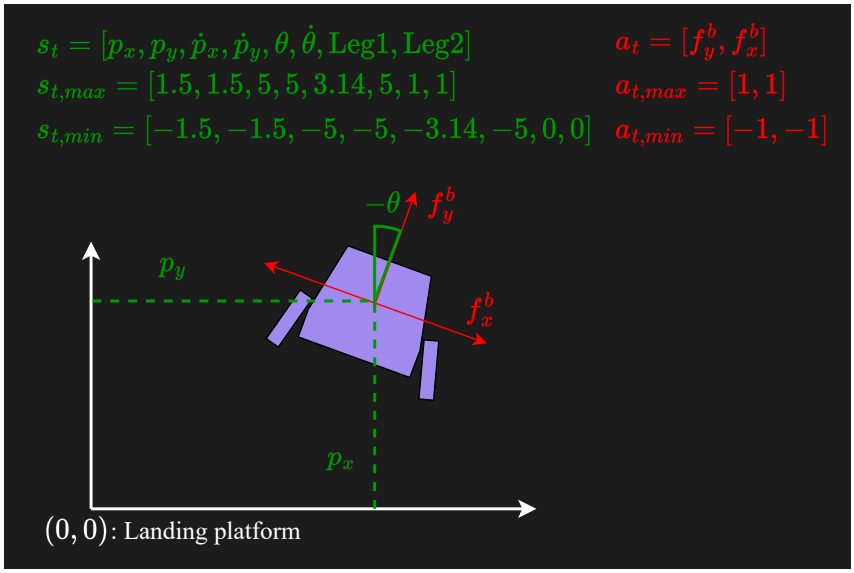


Figure 3.12: This figure illustrates the actions and states of the Lunar Lander Continuous Environment

3.6 Lunar Lander implementation

As briefly mentioned, we test how CES explanations work in practice on the widely used environment Lunar Lander Continuous. The environment has partly discrete and continuous states and a continuous action space. We use this environment because it is well-documented, and since the environment is open source, it makes the results comparable for multiple users of the CES explanation algorithm. Furthermore, having multidimensional, continuous, and discrete states and actions makes it a demanding scenario. Hence if it works well for the lunar lander, it will probably work for simpler environments.

3.6.1 State and action space

The environment has eight states and two actions. Both actions are forces from the body frame from its thrusters, and the states are position, velocity, orientation angle, angular velocity, and two leg sensors. See Figure 3.12. Here we can see all the states' and actions' maximum and minimum values and an illustration of the lander.

The main thruster and first action decide its vertical throttle and have a value between -1 and 1. 1 is the maximum throttle upwards. Although it could be considered misleading, a throttle of -1 is equivalent to turning the main engine off. Because the lunar lander code is borrowed from another framework, the author lacks certainty regarding the reasons behind the scaling. Nevertheless, normalization between -1 and 1 of inputs is often recommended on neural networks [3]. The second action decides whether the lander will rotate and move to the left or right. Here, -1 will rotate and move to the left, and 1 to the right.

In contrast to the actions, all states are measured with respect to a global frame. Here the origin is defined at the landing platform position. Consequently, numerous states with a negative y-directional position are deemed invalid. However, in certain pits located beneath the platform, there could exist valid negative y-positions.

The remaining states are standard except for the angular orientation and leg states. Specifically, the angular orientation is defined as the counterclockwise angle from the global y-axis. Unlike the position and velocity states, the angular orientation state is unscaled (from π to π), and its value can be directly defined in radians without any need for resizing the component.

It might seem confounding that the range of the angle overlaps, where, e.g., an upward orientation can either be 0, π, or $-\pi$. However, this property can prevent angle skipping, as the lander must complete an entire loop to reach the opposite end of the range. The presence of discontinuous jumps, such as from $-\pi/2$ to $\pi/2$, would have complicated the learning process since an actor typically consists of a continuous neural network.

What separates the leg states from the rest of the states is that it is discrete. Both legs are one if it touches the ground and zero if not.

3.6.2 Environment adaption

Unfortunately, the standard gym environments used in Lunar Lander are not made in a way that allows for state alteration. Since our value function depends on it, some modifications to the code are crucial. One of the challenges we must confront is that the environment is wrapped, and we can not, therefore, alternate the code from the gym package. Instead, we find an unwrapped implementation of the Lunar Lander and use that instead. Such code can be found at [38]. Since the code adaption is relatively time-consuming, we do not test CES explanations on multiple environments in this thesis.

To allow for state alteration, we extend the environment's reset function. If we pass in a state as input when resetting the environment, we thwart the terrain alterations and change the state to its predetermined state. As the states are scaled for the agent to learn more efficiently, we need the reset function to scale them back to their actual size. We can see the scaling factors in the step function implementation.

We also need to adapt some aspects of the value function. Since multiple state combinations are invalid, even for states in their allowed range, we cannot use do-calculus regardless. We need to detect that the states are valid before we add them to our value function. Invalid states are detected by checking if the first step in the time frame returns done=True. This approach can cause falsehoods. For example, the step function will return done if the lander landed safely or crashed. Fortunately, such data is rare (it happens once per episode). Since each step is relatively small (up to 250 steps per episode), this has a small likelihood and will not affect the expected values in the value function too much.

3.6.3 Agent

As for the agent, we use the PPO [39] from the stable baselines library [41] with the standard parameters. We use PPO because it achieved better scores than Soft Actor-Critic (SAC) [17] and Deep Deterministic Policy Gradient (DDPG) [24] in the Lunar Lander environment for the tests performed by the author. Since finding the best-suited actor is not the central part of this thesis, we do not investigate further, but the reader should feel free to do so.

CHAPTER

Results 4

The results are presented in chronological order. Hence, we start by showcasing the results from the Time Frame Grouping. Although the end user is interested in the forward explanation, the Time Frame Grouping greatly impacts the end result and is worth inspecting.

We can see the performance for different values of c_D in Figure 4.1. $c_D = 0.5$ should not penalize a dimension increment too much, $c_D = 1$ should penalize it moderately, while $c_D = 4$ should penalize a lot. The number of groupings and hence explanations for $c_D = 0.5$, $c_D = 1$, and $c_D = 4$ are seven, four, and three, respectively. Notice the data from f_y^b has a more dynamic movement, which makes a high-dimensional time frame vector more rewarding.

If a split line from f_x^b is not a split line in f_y^b, then the split is colored in grey for f_y^b and vice versa. Both $c_D = 0.5$ and $c_D = 1$ have an unshared split line. This is intended to clarify how many groups and, therefore, sub-explanations the CES explanation has.

Next, we compare the Time Frame Grouping with and without post-processing in Figure 4.2. The action data is taken from another episode to show the reader various episode data. Without post-processing, we get eleven groups; with post-processing, we get six. None of the split lines is shared for the plot without post-processing, and every except for one is shared with post-processing.

We see the Time Frame Grouping and the belonging CES explanation of an episode in Figures 4.3 and 4.4, respectively. The CES explanation has the parameters given in Table 4.1. We see that the explanation has a complexity limited to human comprehension. The number of causes is between one and two for every time frame, and the effects are between one and ten.

Since no other method comprehensively explains an episode, we can not compare the method in its entirety with another method. Instead, to get a better insight into the correctness of the episodes, we compare iterations of CES explanations with force plots generated with SHAP. Since the methods have different priorities in the explanation, we cannot expect them to give the same result even if both algorithms provide optimal results. Nevertheless, comparing the result with a state-of-the-art method such as SHAP instils assurance in the actual performance of CES explanations. We see the force plots in Figures 4.5 and 4.6. Figures 4.5 show some SHAP explanations consistent with the CES explanations, while Figure 4.6 illustrate SHAP explanations that contradict the CES explanation.

In Figure 4.5, we use the SHAP explanation from the first iteration of the first, third, fourth, and fifth time frame. The first, third, fourth, and fifth SHAP plots strongly agree with the CES explanation, while the second is relatively agreeing. The second force plot indicates that $\dot{\theta}$, x, $\dot{y}$, and θ chronologically are the states that make the lander thrust to the right. Contrarily, the CES explanation only includes x and θ.

In Figure 4.6, we present some SHAP force plots for the first iteration of the second time frames and some force plots that are in the same time frames but not at the first iteration. The explanations deviate in the first iteration in the time frame (first and third plot) but are consistent in iterations 38 and 58.

To showcase the utility of CES explanations, we will compare a CES explanation of a well-trained and poorly trained actor. If the method works well, the user should know what happens during the episode and the agent's considerations. All considerations should be logical for a well-trained agent, but we can expect the opposite for a poorly trained one.

Symbol	Description
c_{act}	0.9
c_{effect}	1.2
c_p^g	1.4
c_{cause}	1.2
c_p^h	0.8

Table 4.1: Parameter value for a complex tuning of Backward Evaluation.

For the agent that is poorly trained, we will find an episode where it gets lucky and lands safely. This compels us to rely solely on the CES explanation representation of the agent's decision-making to determine whether the agent performs its job safely. See its Time Frame Grouping in Figure 4.7 and the CES explanation in Figure 4.8. As we can see, the agent often bases its decision on the velocity in the y-direction. This is not necessarily bad since the agent is initialized relatively stationary in the x-direction just above the platform. That said, it bases the kinetics and kinematics in the x direction on the y velocity, which can indicate that the agent takes uneducated actions.

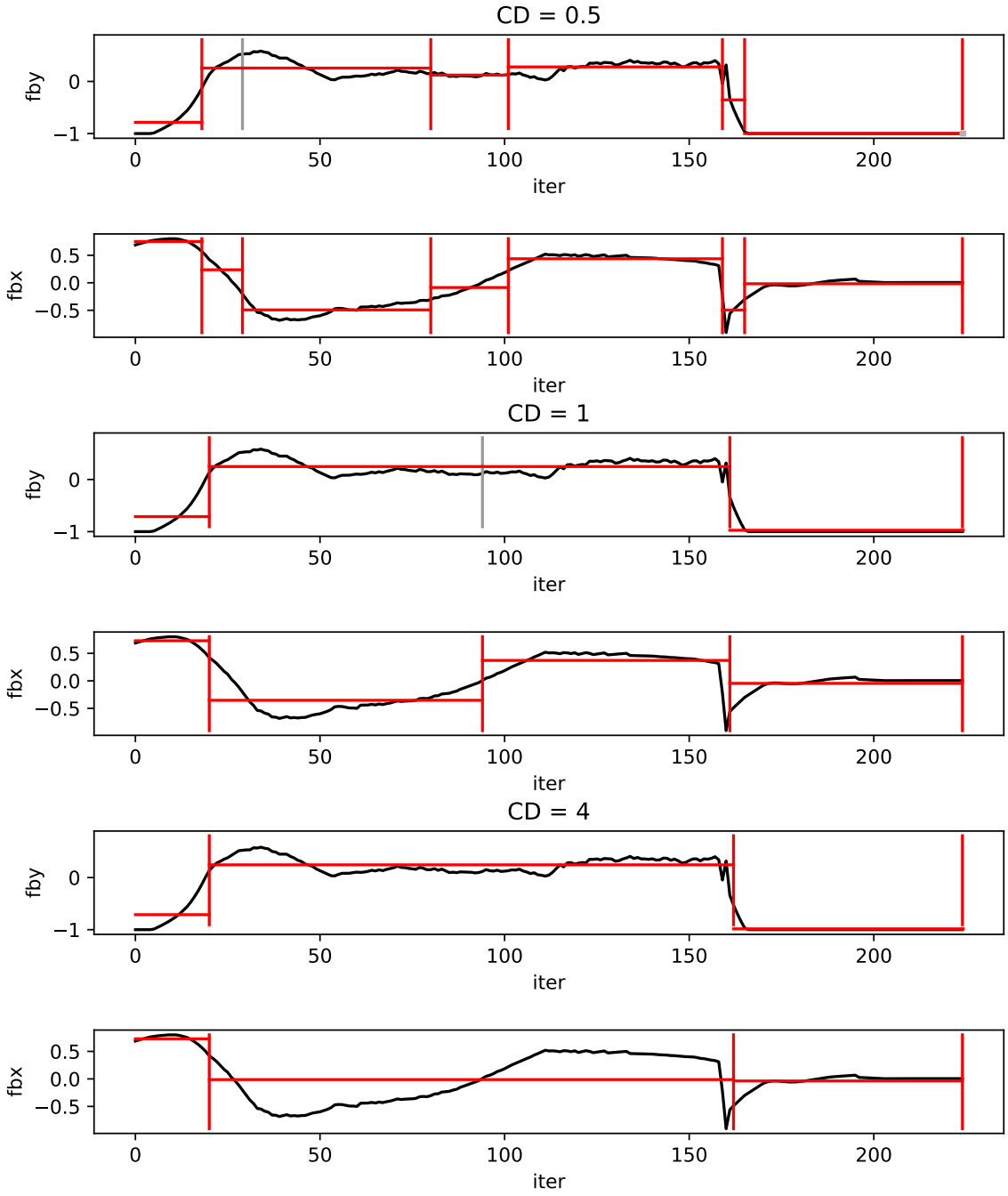


Figure 4.1: Time frames for three different C_D values.

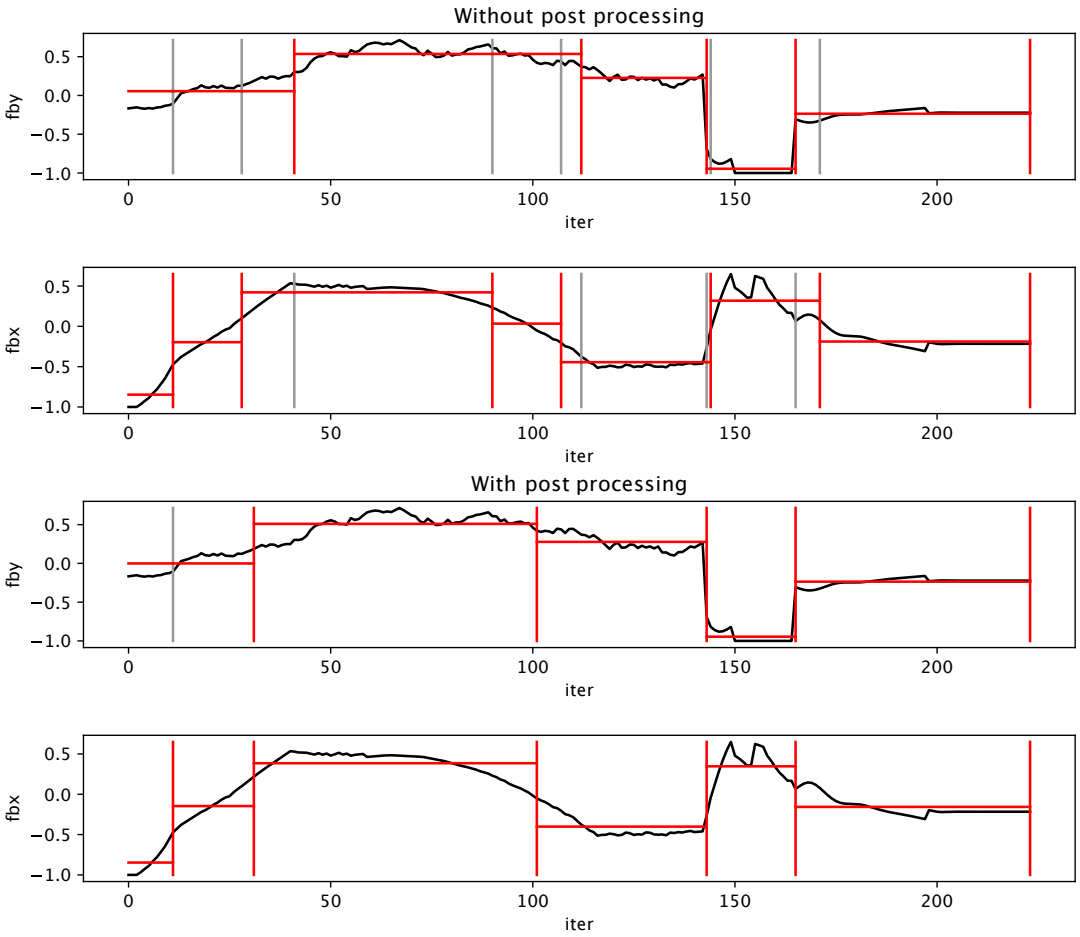


Figure 4.2: Time frames produced with post-processing vs. without.

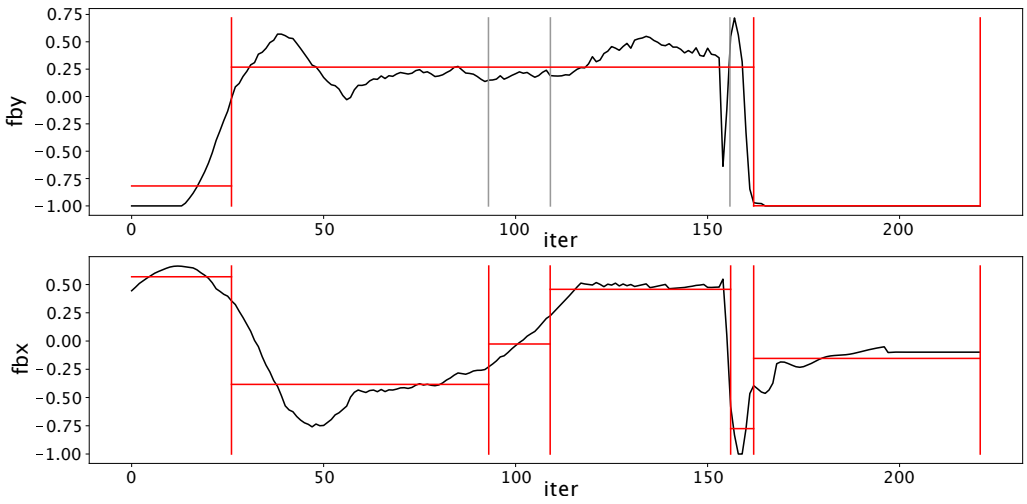
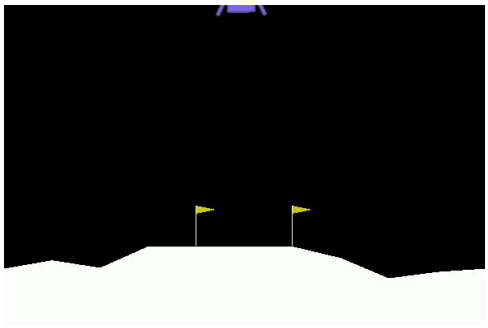


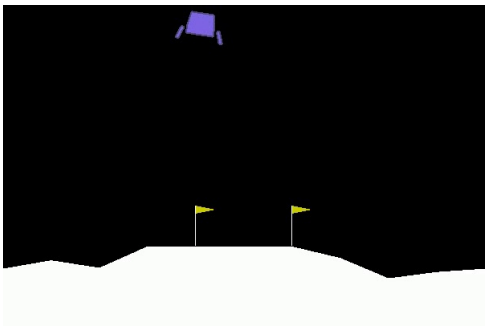
Figure 4.3: Result of Time Frame Grouping

Episode explanation:

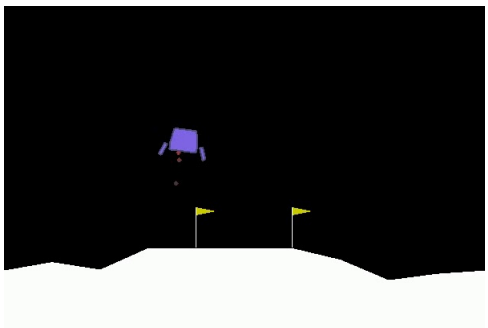
Expected state values: $x = 0.036$, $y = 0.336$, $\dot{x} = 0.025$, $\dot{y} = -0.214$, $\theta = 0.003$, $\dot{\theta} = 0.001$, Leg1 $= 0.4374$, Leg2 $= 0.441$
Expected action values: $f_x^b = -0.096$, $f_y^b = -0.346$



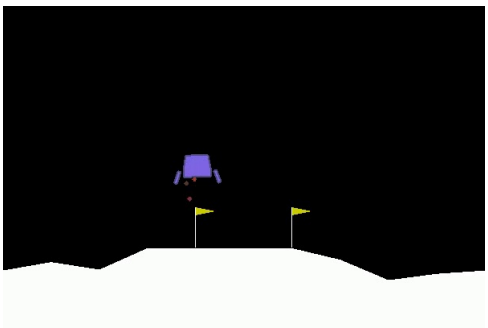
Time frame: 0-26
State: $x = -0.006$, $y = 1.414$, $\dot{x} = -0.702$, $\dot{y} = 0.141$, $\theta = 0.008$, $\dot{\theta} = 0.159$, Leg1 $= 0$, Leg2 $= 0$
Cause: Do $f_x^b = 0.568$ because $\dot{x} = -0.702$
Effect: $f_x^b = 0.568$ contributes to $\theta = -0.134$ in iter 26



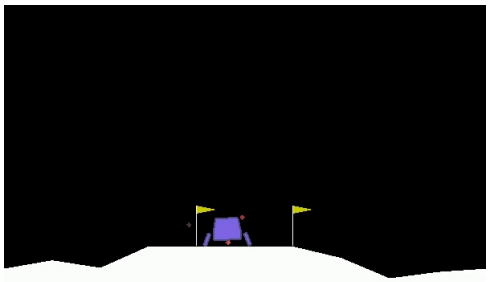
Time frame: 26-93
State: $x = -0.173$, $y = 1.286$, $\dot{x} = -0.580$, $\dot{y} = -0.552$, $\theta = -0.134$, $\dot{\theta} = -0.328$, Leg1 $= 0$, Leg2 $= 0$
Cause 1: Do $f_y^b = 0.268$ because $\theta = -0.134$
Cause 2: Do $f_x^b = -0.383$ because $\theta = -0.134$
Effect 1: $f_y^b = 0.268$ contributes to $\dot{y} = -0.423$, Leg2 $= 0$ in iter 93, $x = -0.198$, $\theta = 0.021$ in iter 109, $\dot{\theta} = -0.728$ in iter 156
Effect 2: $f_x^b = -0.383$ contributes to $x = -0.198$, $\theta = 0.021$ in iter 109, $\dot{\theta} = -0.728$ in iter 156



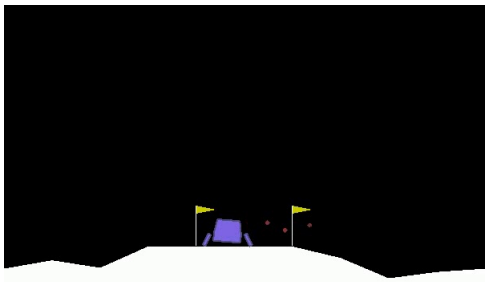
Time frame: 93-109
State: $x = -0.256$, $y = 0.568$, $\dot{x} = 0.308$, $\dot{y} = -0.424$, $\theta = -0.136$, $\dot{\theta} = 0.182$, Leg1 $= 0$, Leg2 $= 0$
Cause: Do $f_y^b = 0.268$ because $\dot{y} = -0.424$, Leg2 $= 0$
Effect: $f_y^b = 0.268$ contributes to $x = -0.198$ in iter 109, $\dot{\theta} = -0.728$ in iter 156



Iteration: 109-156
State: $x = -0.198$, $y = 0.408$, $\dot{x} = 0.384$, $\dot{y} = -0.448$, $\theta = 0.021$, $\dot{\theta} = 0.219$, Leg1 $= 0$, Leg2 $= 0$
Cause: Do $f_x^b = 0.458$ because $x = -0.198$, $\theta = 0.021$
Effect: $f_x^b = 0.458$ contributes to $\dot{\theta} = -0.728$ in iter 156

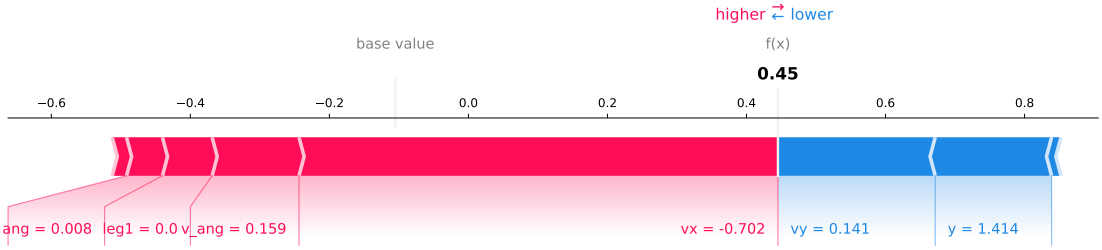


Time frame: 156-162
State: $x = -0.075$, $y = 0.002$, $\dot{x} = 0.131$, $\dot{y} = -0.239$, $\theta = 0.097$, $\dot{\theta} = -0.728$, Leg1 $= 0$, Leg2 $= 1$
Cause 1: Do $f_x^b = -0.775$ because $\dot{\theta} = -0.728$
Cause 2: Do $f_y^b = 0.268$ because $\dot{\theta} = -0.728$
Effect 1: $f_x^b = -0.775$ contributes to $\dot{y} = -0.054$ in iter 162, Leg1 $= 1$, Leg2 $= 1$ in iter 221
Effect 2: $f_y^b = -0.268$ contributes to $\dot{y} = -0.054$ in iter 162, Leg2 $= 1$ in iter 221

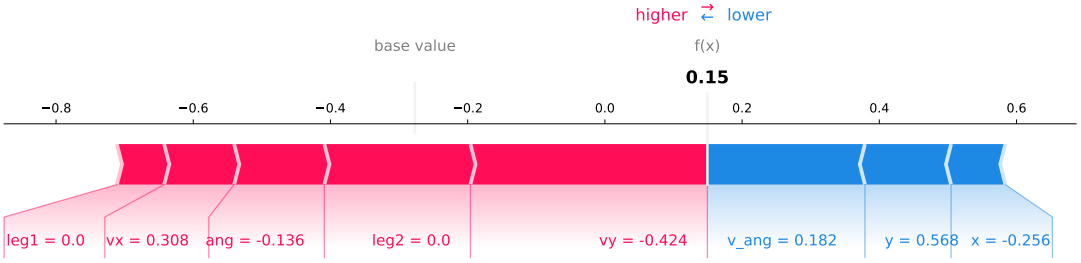


Iteration: 162-221
State: $x = -0.071$, $y = -0.015$, $\dot{x} = 0.058$, $\dot{y} = -0.054$, $\theta = -0.077$, $\dot{\theta} = -0.062$, Leg1 $= 1$, Leg2 $= 1$
Cause 1: Do $f_x^b = -0.154$ because $\dot{y} = -0.054$
Cause 2: Do $f_y^b = -0.999$ because $\dot{y} = -0.054$
Effect 1: $f_x^b = 0.154$ contributes to $x = -0.066$ $y = -0.001$, $\dot{x} = 0.000$, $\theta = -0.002$ $\dot{\theta} = -0.000$, Leg1 $= 1$, Leg2 $= 1$ in iter 221
Effect 2: $f_y^b = -0.999$ contributes to $y = -0.001$, $\dot{x} = 0.000$, $\dot{y} = 0.000$ in iter 221

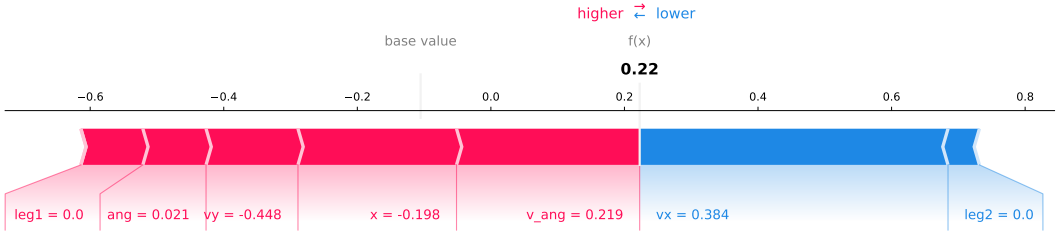
Figure 4.4: CES explanation of a well-trained agent with the Backward evaluation parameters given in Table 4.1



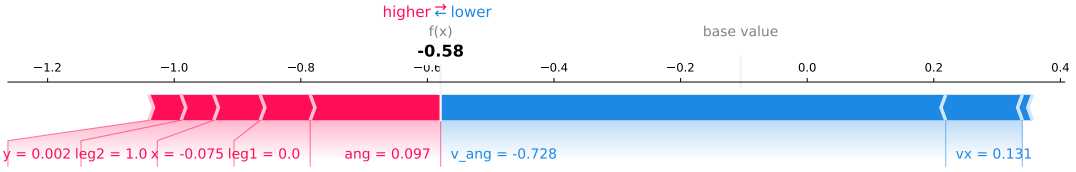
(a) Iteration 0. CES explanation: Do $f_x^b = 0.568$ because $\dot{x} = -0.702$



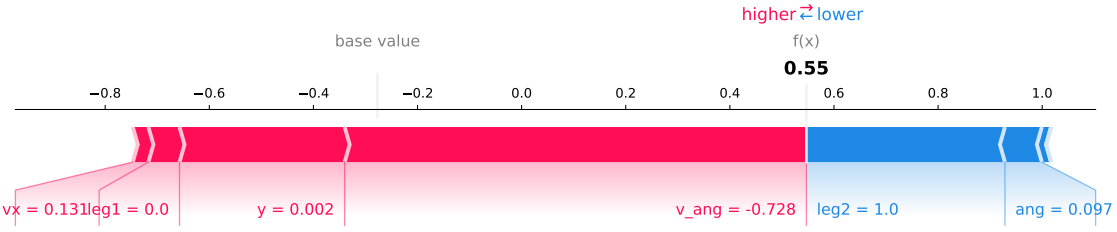
(b) Iteration 93. CES explanation: Do $f_y^b = 0.268$ because $\dot{y} = -0.424$, Leg2 $= 0$



(c) Iteration 109. CES explanation: Do $f_x^b = 0.458$ because $x = -0.198$, $\theta = 0.021$

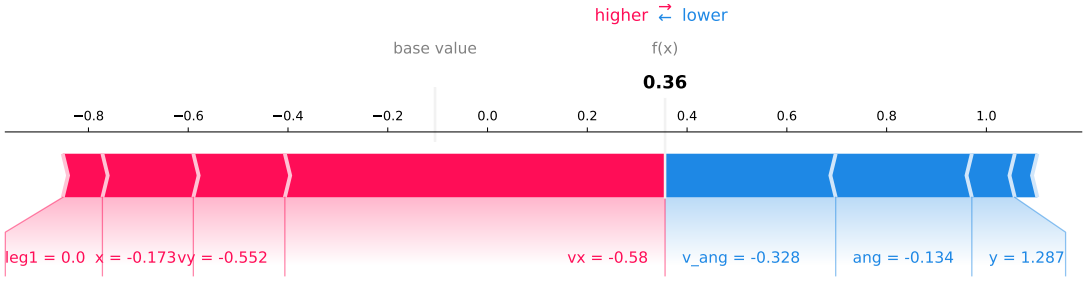


(d) Iteration 156. CES explanation: Do $f_x^b = -0.775$ because $\dot{\theta} = -0.728$

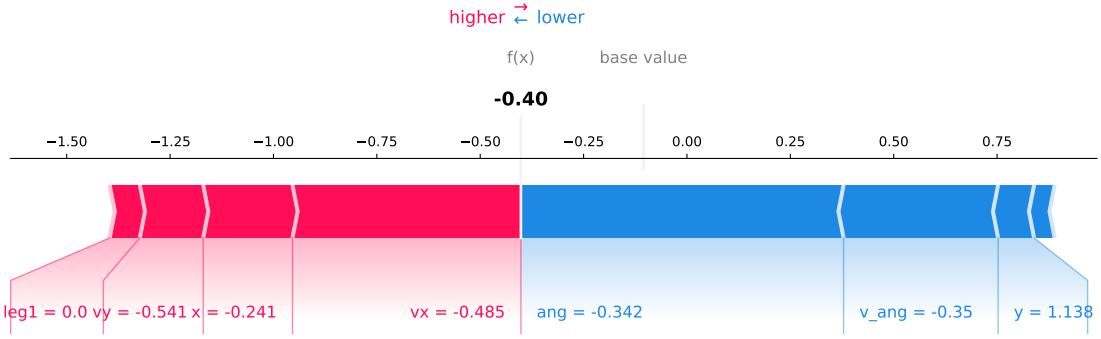


(e) Iteration 156. CES explanation: Do $f_y^b = 0.268$ because $\dot{\theta} = -0.728$

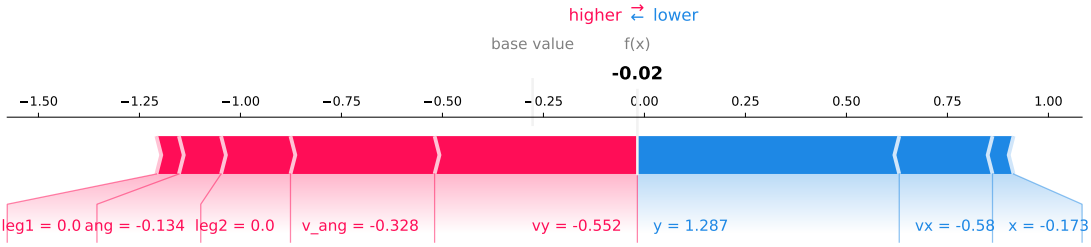
Figure 4.5: Compare CES explanation of an agent-iteration with SHAP. Both explanations are consistent in these cases.



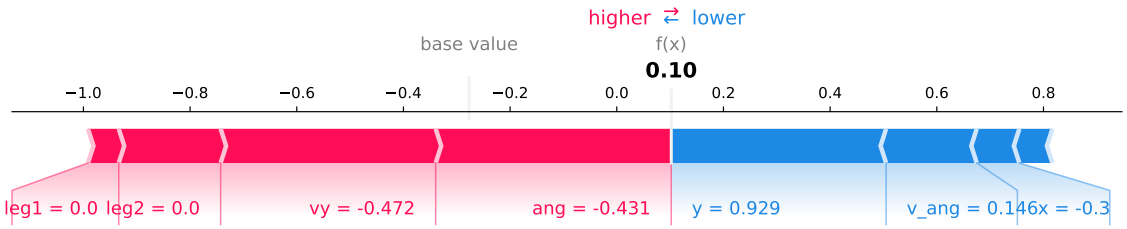
(a) Iteration 26. CES explanation: Do $f_x^b = -0.383$ because $\theta = -0.134$



(b) Iteration 38. CES explanation: Do $f_x^b = -0.383$ because $\theta = -0.134$



(c) Iteration 26. CES explanation: Do $f_y^b = 0.268$ because $\theta = -0.134$



(d) Iteration 58. CES explanation: Do $f_y^b = 0.268$ because $\theta = -0.134$

Figure 4.6: Compare CES explanation of an agent-iteration with SHAP in time frame two. The first and third explanations are not consistent.

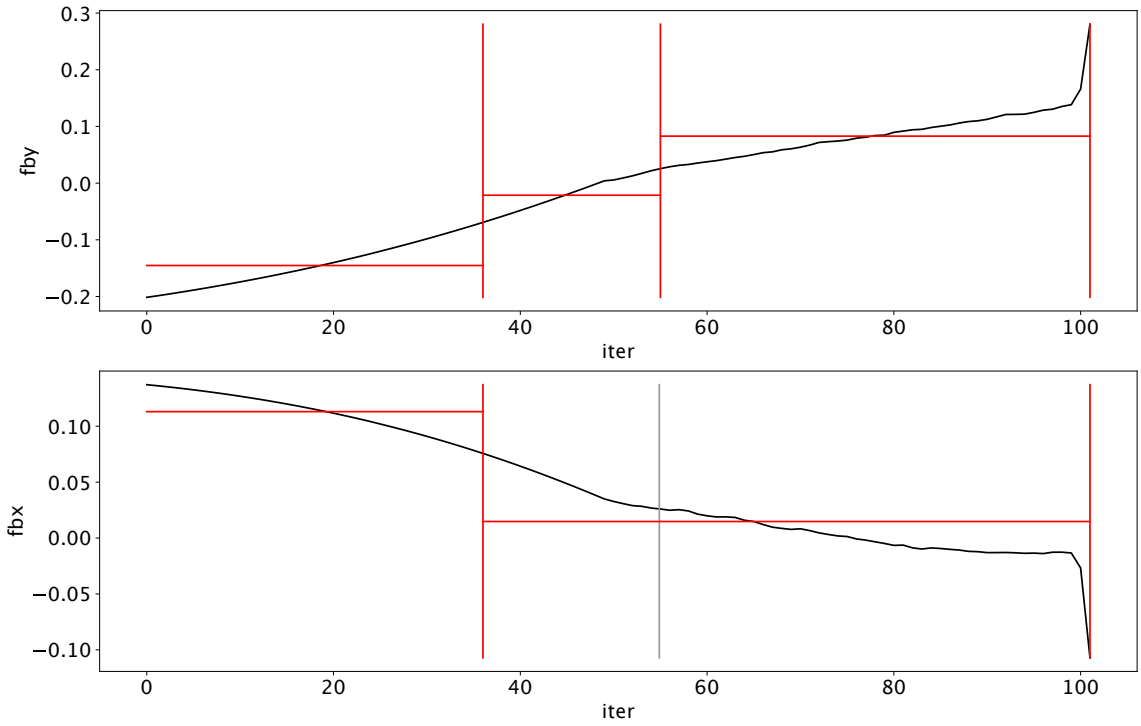
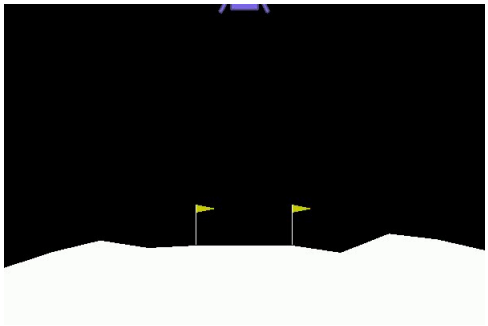


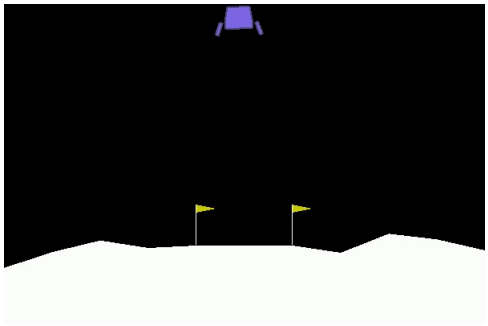
Figure 4.7: Time frames from the poorly trained agent

Episode explanation:

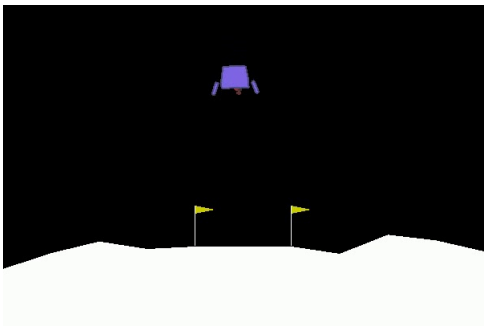
Expected state values: $x = -0.109$, $y = 0.436$, $\dot{x} = -0.046$, $\dot{y} = -0.206$, $\theta = 0.034$, $\dot{\theta} = -0.002$, Leg1 $= 0.381$, Leg2 $= 0.392$
Expected action values: $f_x^b = 0.206$, $f_y^b = -0.147$



Time frame: 0-36
State: $x = -0.001$, $y = 1.419$, $\dot{x} = -0.064$, $\dot{y} = 0.372$, $\theta = 0.001$, $\dot{\theta} = 0.014$, Leg1 $= 0$, Leg2 $= 0$
Cause 1: Do $f_y^b = -0.145$ because $\dot{x} = -0.064$, $y = 1.419$
Cause 2: Do $f_x^b = 0.113$ because $\dot{y} = -0.063$
Effect 1: $f_y^b = -0.145$ contributes to $\dot{y} = 0.372$ in iter 36
Effect 2: $f_x^b = 0.113$ contributes to $\dot{y} = 0.372$ in iter 36



Time frame: 36-55
State: $x = -0.023$, $y = 1.321$, $\dot{x} = -0.064$, $\dot{y} = -0.588$, $\theta = 0.026$, $\dot{\theta} = 0.014$, Leg1 $= 0$, Leg2 $= 0$
Cause 1: Do $f_y^b = -0.021$ because $\dot{y} = -0.588$
Cause 2: Do $f_x^b = 0.015$ because $\dot{y} = -0.588$
Effect 1: $f_y^b = 0.268$ contributes to $\dot{y} = -0.971$ in iter 55, $x = -0.082$, $y = -0.020$, $\dot{x} = -0.140$, $\dot{y} = -0.971$, $\theta = 0.093$, $\dot{\theta} = -0.321$, Leg1 $= 1$, Leg2 $= 1$ in iter 101
Effect 2: $f_x^b = 0.015$ contributes to $x = -0.082$, $y = -0.020$, $\dot{x} = -0.140$, $\dot{y} = -0.971$, $\theta = 0.093$, $\dot{\theta} = -0.321$, Leg2 $= 1$ in iter 101



Time frame: 55-101
State: $x = -0.035$, $y = 0.966$, $\dot{x} = -0.077$, $\dot{y} = -0.971$, $\theta = 0.041$, $\dot{\theta} = 0.008$, Leg1 $= 0$, Leg2 $= 0$
Cause 1: Do $f_x^b = 0.015$ because $\dot{y} = -0.971$
Cause 2: Do $f_y^b = 0.082$ because $\dot{y} = -0.971$
Effect 1: $f_x^b = 0.268$ contributes to $x = -0.082$ in iter 101
Effect 2: $f_y^b = 0.082$ contributes to $x = -0.082$, $y = -0.020$, $\dot{y} = -0.971$, $\theta = 0.093$, Leg1 $= 1$, Leg2 $= 1$ in iter 101

Figure 4.8: CES explanation of a poorly trained agent

CHAPTER

Discussion 5

5.1 Time frames

When deciding the constant c_D, it is hard not to draw any subjective conclusions. Here we need to balance between accuracy and simplicity of the explanation. See Figure 4.1. Having $c_D = 0.5$ is, without a doubt, the most precise explanation. Especially when it comes to f_x^b, we decrease the overall variance by having a time frame vector size of seven instead of four or three. The gradual movement of the f_x^b graph between iterations 25 and 110 and the drop at iteration 155 is well captured. On the other hand, f_x^b seems to be overly complex. For f_y^b from iterations 25 to 155, the data is more or less the same, but the algorithm still divides it into three time frames.

There is generally higher variance in the time frames for $c_D = 1$ in the f_x^b graph. That said, the grouping is reasonably accurate, and the time frame dimensionality is three lower than for $c_D = 0.5$. It produces a simplified explanation of f_y^b without considerably compromising accuracy. On the unfortunate side, the algorithm does not capture the drop in f_x^b at iteration 155.

$c_D = 4$ gives a similarly simple f_y^b Time Frame Grouping as $c_D = 1$. The split lines are more or less positioned in the same place, so we will not add any additional remarks. In f_x^b, the grouping could be better. Although there only are three time frames, the mean of the last two neighbour time frames is almost equal. Hence joining these two would probably give a lower loss. The reason why we have that additional split line is likely because of the post-processing.

Because of the dimensionality reduction, we can guarantee that the algorithm tried removing the second split but found that the loss increased by doing so. This might seem strange, but the time frame it compares differs somewhat from the figure. The time frames that are plotted are merged after dimensionality reduction. Therefore the second and third-time frames might have other ranges with lower variance when sent through the dimensionality reduction function. This raises the question of whether the post-processing negatively impacts the result.

As we can see in Figure 4.2, this is certainly not the case for the example episode. The post-processing first removed a dimension from f_x^b and continued by merging all frames except one, which resulted in six groups. We see that eleven explanations are overly complex and that six fit the data considerably better.

We use $c_D = 1$ with the post-processing applied for the upcoming tests since this generally gives the best results.

5.2 CES explanation validation

We start by evaluating the first time frame building ourselves up to the last. See Figure 4.4. It is recommended to watch the video of the episode in parallel with the CES explanation, although it is not a necessity. The video can be found as an attachment on the submission page.

Iteration 0-26: The lander is initialized right above the platform, with a force pointing in a random direction. In this case, the force made our lander speed to the left and downward. The CES explanation evaluated the horizontal kinetics as most relevant in this time frame. Hence, it

explains that it corrects its left speed with a right force.

Generally, f_x^b makes the lander move left or right and turn accordingly because of its upward off-centred side-thrusters' positions. Although the lander accelerates, moving from the left to the right, the explanation of the action effects only focuses on the lander's rotation. This is most likely because the lander can only reduce its speed slightly before it rotates so much that it has to correct itself.

Iteration 26-93: The agent's vertical thruster helps the lander to move towards the landing platform since it has an orientation to the right. We, therefore, see that both f_x^b and f_y^b are a part of the episode explanation. For the lander to fluently rotate to the left, it uses a thrust upwards and to the left. We can argue that the x-position should have been included as a cause of f_y^b in that a thrust in f_y^b makes the lander to move horizontally toward the landing platform. The exclusion of the x-position in the CES explanation might stem from two distinct reasons.

Firstly, it can be because of the CES explanation itself. Either because the probabilistic objective sampling did not find the optimal solution of the objective or because adding the sub-explanation resulted in more explanation complexity than accuracy.

Secondly, it can be because of the agent. The x-position of the agent may be de-prioritized in that it is still high above the ground. Although the action positively affects the x-position, it may be more of a result of the action than an actual assessment.

One of the effects of f_y^b is to gain sufficient altitude so the second leg does not touch the ground. Because the lander is positioned so that only the left (second) leg will hit the ground outside the platform, it is prioritized in the explanation. Both f_x^b and f_y^b help the lander to neutralize the orientation and move towards the landing platform. Additionally, the actions produce an angular velocity that is considered and compensated for by the agent in iteration 156.

Iteration 93-109: Next up, the lander finds the velocity downwards and the left leg's state relevant when applying a high vertical force. This is most likely because the agent has a low altitude and wants to dampen the downward velocity to get a smooth landing. The agent's slight tilt to the left makes f_y^b affect the x-position in the upcoming time frame.

Surprisingly, it also affects the angular velocity in iteration 156. f_y^b will affect the rotation speed somewhat because the vertical thruster is distant from the mass center of the lander. Generally, f_x^b has a higher effect, but not here since f_x^b is close to zero. Having f_x^b relatively neutral might make f_y^b the highest impacting force. See the third time frame in Figure 4.3 to justify the statement. As always, it can alternatively be because the probabilistic objective sampling did not find the optimal solution.

Iteration 109-156: In the fourth time frame, the lander applies thrust to the right because of its slight tilt to the left and to get closer to the center of the platform. We will go into more detail when comparing the explanation with SHAP, but the agent probably assesses the angular velocity the most. Still, this state is possibly de-prioritized to keep the explanation simple. The effect of $f_x^b = 0.458$ inverts the angular velocity from turning left to right in iteration 156.

Iteration 156-162: The agent applies a high force to the left to compensate for the angular velocity from the previous time frame and make the landing soft. The force lasts for a short period and has a high amplitude to get a quick response. It also applies an upward force for the same reason. These two actions have a high effect on the leg states and the vertical speed of the lander.

Iteration 162-221: Finally, the lander makes minor adjustments with a slight trust to the left and with no thrust from the vertical engine. The agent takes this action based on the y direction's low speed. One can argue that several states are relevant in the decision that is not mentioned as a cause. If we assume this is the case, we should further inspect whether the agent acts safely with similar state conditions. This can be done by initializing the agent in the suspicious state area and observing how well the agent performs here. We run it in this area multiple times, and if the agent performs poorly, we can train it based on the unsatisfactory episode outcomes. Since the speed is low and the agent is well on its way to landing safely, the current time frame is unlikely to create a dangerous situation.

The effects of the actions are quite a few. This might seem strange since the actions have a low impact on the state's change. The reason possibly is because the lander has a relatively low speed. This makes the forces applied by the actions more prominent than all the velocity states to the state transitions. Equally important, the final nodes are constrained to a part of the explanation, and the more states that require an explanation, the more action effects can be expected.

5.2.1 Compare CES explanation with SHAP

To validate parts of the explanation further, we compare it with SHAP. We do this on the same episode as above for different iterations of the agent. Something to be aware of is that the methods have different strategies to explain the feature relevance. A feature may highly respond to the output when its relevancy is compared with a specific subset, but it responds less to the output when examining the feature in others. SHAP will take the weighted average of all the subsets of a feature, while CES explanation looks for the most explaining subset. CES also sometimes compromises the accuracy of the explanation to keep it simple.

In Subfigures 4.5a, 4.5b, 4.5d, and 4.5e, we see that the CES explanation corresponds well with SHAP. One distinct feature clearly has the highest effect in the force plot, which is the same feature as the CES explanation mentions. Hence for these specific cases, there is little to discuss.

On the other hand, Subfigure 4.5c prioritizes the input features differently. Here, the CES explanation mentions the features with the second and fourth highest positive signed SHAP values, while the first and third are ignored. As discussed earlier, this can have multiple reasons. It might be that the combination of the x-position and the angle describes the output the most, but for every other subset of inputs, it has a low influence on the output. It might also be that the solver did not discover the optimal solution or that the most accurate explanation is too complex. When that is said, the angle and the x-position describe relatively well why f_x^b is positive in the force plot.

Figure 4.6 focuses on the two actions from the second time frame. The second time frame illustrates a special case where the SHAP values for the iteration at the start of the time frame disagree with the CES explanation. The results are more agreeable if we compare the SHAP values for a later iteration in the same time frame. See Subfigures 4.6b and 4.6d. We can assume this is due to the action's progression in the time frame and the strategy to decide which iteration we want to explain. The second time frame (see Figure 4.3) has very different action values at the start versus the ending. Furthermore, the states and associated state priorities of the agent change significantly throughout the time frame.

We can take two measures to reduce the number of such cases. Foremost, we can prevent such time frames from being created in the first place. A quick fix is to increase the number of time frames (decrease c_D). Then, the variations internally in the time frame reduce, but the explanation complexity grows. We can alternatively change the time frame objective to group the actions into linear growth instead of constant values. I.e., increase f_x^b gradually from 0.4 to 0.6 because $x = 0.7$. If the actions change rapidly, we can gain a lot of accuracy by allowing such explanations. When that is said, this type of explanation adds one extra layer of explanation complexity.

Additionally, we can use a more robust way to decide which inputs we want to explain in the time frame. By now, the CES explanation only explains the state input in the time frame that gives the output closest to the mean action. This input is not necessarily the mean state for the time frame. Consequently, the CES explanation can be based on unrepresentative state combinations.

We would get more robust explanation results if we calculate the value functions of all states in the time frame and take the average. That said, this would drastically increase the running time. As the running time is already high, we would be better off taking a different approach. Instead, we can take the weighted average of a few representative states. The more representative the state is, the higher weight in the average it gets. We can achieve this by utilizing the Time Frame Grouping on the states (not just the actions). Then explain the states from each time frame, and weigh them according to their belonging time frame lengths.

5.3 CES explanation for a poorly trained agent

At first sight, we see that the explanation for the poorly trained agent is shorter than for the well-trained agent. Both in terms of iterations and time frames. See Figure 4.7. Having fewer iterations indicates that the agent landed faster, which is good. That said, this has more to do with the agent's initial condition than the agent itself. We wanted to explain an episode where the poorly trained agent lands safely. This way, multiple indications of the agent's awareness are required, surpassing the mere outcome of the episode. Hence in contrast to the well-trained agent, the poorly trained is initialized with a lower speed in the x-direction (easier state conditions). Because this is a more straightforward scenario, the agent can land faster without the same maneuvering difficulties.

Slowly evolving actions cause fewer time frames. As the actions are so static, we needed to decrease c_D to get more than one time frames to begin with. We can calculate the time frame parameters based on the magnitude of the actions and the episode length to make this process automatic.

The agent generally provided small actions, also for episodes that required tight maneuvers. Therefore, the low magnitude is not solely due to the agent's simple initial conditions. We can assume that small action changes have to do with how the agent's neural network is initialized. More specifically, the agent is initialized with weights that give low variances of the outputs even for highly changing inputs.

As briefly mentioned in the Results, the poorly trained agent strongly relies its decision-making on the y-velocity. See Figure 4.8. Since the agent is always oriented upward, the body frame and the global frame have similar directions on the axes. This makes the kinetics and kinematics for x and y close to independent. Despite this, the agent bases f_x^b on the velocity in the y-direction instead of, for example, mainly focusing on its x-position. Although several other actions seem illogical, this is the strongest indication that the agent is unreliable.

CHAPTER

Conclusion and Future Work 6

This thesis aims to invent a method that comprehensively explains how a Reinforcement Learning agent acts in an episode. Furthermore, the method seeks to clarify which states the agent bases its condition on, to learn from the agent, or evaluate if the agent can be trusted. The author's search yielded no alternative methods to the problem. As Reinforcement Learning advances, we expect RL to be applied in more high-risk applications. Consequently, we need an equivalent level of trust in the agent.

Based on the explanation provided by the Time Frame Grouping, the overall CES explanations, and the SHAP comparison, the results of the method are beyond the author's expectations. A method typically improves with incremental steps, and since this method introduces a solution to an unexplored problem (comprehensive episode explanations), extraordinary results can not be expected. If CES explanations are deployed for a vast number of problems, we can easily detect its strengths and weaknesses and propose improvements.

At first, it may be advisable to apply the algorithm to multiple control applications. Control applications primarily involve continuous states and actions with a small dimension size compared to, for example, chess. Given the high risks of controlling a physical system, obtaining a comprehensive understanding of the agent's situational awareness can significantly enhance trustworthiness.

The method can possibly also be specialized for discrete and high-dimensional state and action spaces. By capitalizing on the concepts while tailoring the methodology, we may elucidate the mechanisms behind the success of world-class chess agents and understand the Reinforcement Learning-based parts of a language model. We can expect many more practical applications as the Reinforcement Learning capabilities increase.

6.1 Future work

Given that CES explanations are novel, various adaptations could potentially yield improved results. This section elucidates a selection of such modifications and deliberates on their potential enhancements.

We start by addressing the robustness of the explanations. Since we find the influential edges recursively, the variations in the result will build up over time. This means we can get significantly different outcomes by making minor adjustments. There are often several ways to explain a situation, and we can, therefore, argue that the outcome variance is not a big deal. That said, having more stable outcomes makes it easier to find improvements in the algorithm and recreate the results of interests.

To enhance consistency, we propose a global solver instead of recursively solving each iteration. By applying a methodology similar to probabilistic objective sampling on the entire graph simultaneously, we can achieve more reliable outcomes. Given the vast decision space, a thorough search involving numerous iterations becomes necessary. However, the time required to evaluate a single iteration is relatively short when all the value functions are already determined. An important consideration is the memory needed to store the value function's return for all possible subsets

associated with each node. The algorithm may require hardware with high memory specifications to run.

Alternatively, we can find the $\mathcal{I}_t^{effect}$, $\mathcal{I}_t^{act}$ and $\mathcal{I}_t^{cause}$ simultaneously. This procedure may not have the same result consistently as solving the entire graph simultaneously but deliberates with another method enhancement. As for now, $\mathcal{I}_t^{cause}$ considers $\mathcal{I}_t^{effect}$ when finding a simple but still accurate explanation, but $\mathcal{I}_t^{effect}$ does not consider $\mathcal{I}_t^{cause}$. Instead of exclusively penalizing the nodes in $\mathcal{I}_t^{cause}$ that do not belong to $\mathcal{I}_t^{effect}$, it is advisable to penalize the inverse scenario uniformly. In more mathematical terms, $\left|\mathcal{I}_t^{cause} \setminus \mathcal{I}_t^{effect}\right|$ should be equally penalised as $\left|\mathcal{I}_t^{effect} \setminus \mathcal{I}_t^{cause}\right|$. This way, we will minimize $|\mathcal{O}_t|$ and not just an approximation of it (given that $c_{cause} = c_{effect}$). In other words, we will make the explanation compromises of the cause and effect nodes more equally balanced.

Something to be aware of is that $\mathbb{E}[f(\boldsymbol{X})] \neq f(\mathbb{E}[\boldsymbol{X}])$. This means we can not necessarily explain an input value in the following manner: "Because the input is higher than its expected value, the output is lower than its expected value." We can only guarantee the statement is correct if the function f strictly increases, decreases, or is constant. Hence the only type of explanation we can draw for certain is: "Because of the input's specific value, the output is lower than its expected value." That is why SHAP only compares its output value with the expected output, while it does not compare the input.

If we want to compare two input values and explain how they correlate to the output, we can only do this for a local area. Both LIME and counterfactual find explanations in a local area. Hence if we were to use an inspired approach to LIME or counterfactuals when finding influential edges, it might allow for these types of explanations.

Finally, it might be an idea to experiment with the methods suggested in subsection 5.2.1 regarding the misleading results from the second time frame.

Bibliography

[1] NIH National Institute on Aging (NIA). *Placebos in Clinical Trials*. en. URL: https://www.nia.nih.gov/health/placebos-clinical-trials (visited on 2nd May 2023).

[2] *AlphaZero: Shedding new light on chess, shogi, and Go*. en. URL: https://www.deepmind.com/blog/alphazero-shedding-new-light-on-chess-shogi-and-go (visited on 24th Apr. 2023).

[3] baeldung. *Normalizing Inputs of Neural Networks — Baeldung on Computer Science*. en-US. July 2020. URL: https://www.baeldung.com/cs/normalizing-inputs-artificial-neural-network (visited on 29th May 2023).

[4] Reza Bagheri. *Introduction to SHAP Values and their Application in Machine Learning*. Medium. 8th Aug. 2022. URL: https://towardsdatascience.com/introduction-to-shap-values-and-their-application-in-machine-learning-8003718e6827 (visited on 28th Apr. 2023).

[5] Brady Neal - Causal Inference. *2 - Potential Outcomes (Week 2)*. Sept. 2020. URL: https://www.youtube.com/watch?v=5x_pPemAVxs (visited on 1st May 2023).

[6] *Causal Inference - an overview — ScienceDirect Topics*. URL: https://www.sciencedirect.com/topics/social-sciences/causal-inference (visited on 2nd May 2023).

[7] Karam Daaboul. *Reinforcement Learning: Dealing with Sparse Reward Environments*. en. Aug. 2020. URL: https://medium.com/@m.k.daaboul/dealing-with-sparse-reward-environments-38c0489c844d (visited on 29th May 2023).

[8] *Discovering novel algorithms with AlphaTensor*. en. URL: https://www.deepmind.com/blog/discovering-novel-algorithms-with-alphatensor (visited on 24th Apr. 2023).

[9] Kathryn A. Dowsland and Jonathan M. Thompson. 'Simulated Annealing'. In: *Handbook of Natural Computing*. Ed. by Grzegorz Rozenberg, Thomas Bäck and Joost N. Kok. Berlin, Heidelberg: Springer, 2012, pp. 1623–1655. ISBN: 978-3-540-92910-9. DOI: 10.1007/978-3-540-92910-9_49. URL: https://doi.org/10.1007/978-3-540-92910-9_49 (visited on 2nd May 2023).

[10] Nadav Dym. *Quasi Branch and Bound for Smooth Global Optimization*. 27th May 2020. DOI: 10.48550/arXiv.2005.13728. arXiv: 2005.13728[math]. URL: http://arxiv.org/abs/2005.13728 (visited on 2nd May 2023).

[11] Stefan Endres, Carl Sandrock and Walter Focke. 'A simplicial homology algorithm for Lipschitz optimisation'. In: *Journal of Global Optimization* 72 (Oct. 2018). DOI: 10.1007/s10898-018-0645-y.

[12] Jacob Feldman. 'The simplicity principle in perception and cognition'. In: *Wiley interdisciplinary reviews. Cognitive science* 7.5 (Sept. 2016), pp. 330–340. ISSN: 1939-5078. DOI: 10.1002/wcs.1406. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5125387/ (visited on 26th May 2023).

[13] Timo Freiesleben. 'The Intriguing Relation Between Counterfactual Explanations and Adversarial Examples'. In: *Minds and Machines* 32.1 (1st Mar. 2022), pp. 77–109. ISSN: 1572-8641. DOI: 10.1007/s11023-021-09580-9. URL: https://doi.org/10.1007/s11023-021-09580-9 (visited on 2nd May 2023).

[14] Dimitra Giannakopoulou, Kedar S. Namjoshi and Corina S. Păsăreanu. 'Compositional Reasoning'. en. In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke et al. Cham: Springer International Publishing, 2018, pp. 345–383. ISBN: 978-3-319-10575-8. DOI: 10.1007/978-3-319-10575-8_12. URL: https://doi.org/10.1007/978-3-319-10575-8_12 (visited on 26th May 2023).

[15] Clark Glymour, Kun Zhang and Peter Spirtes. 'Review of Causal Discovery Methods Based on Graphical Models'. In: *Frontiers in Genetics* 10 (2019). ISSN: 1664-8021. URL: https://www.frontiersin.org/articles/10.3389/fgene.2019.00524 (visited on 2nd May 2023).

[16] Riccardo Guidotti. 'Counterfactual explanations and how to find them: literature review and benchmarking'. en. In: *Data Mining and Knowledge Discovery* (Apr. 2022). ISSN: 1573-756X. DOI: 10.1007/s10618-022-00831-6. URL: https://doi.org/10.1007/s10618-022-00831-6 (visited on 26th May 2023).

[17] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. arXiv:1801.01290 [cs, stat]. Aug. 2018. DOI: 10.48550/arXiv.1801.01290. URL: http://arxiv.org/abs/1801.01290 (visited on 24th Apr. 2023).

[18] Eligius Hendrix. 'Global optimization at work.' Journal Abbreviation: Dr. P. van Beek, ir. A.J.M. Beulens (supervisors) Wageningen Agricultural University, The Netherlands (1998) 248 pp. ISBN 90-5485-874-5. Publication Title: Dr. P. van Beek, ir. A.J.M. Beulens (supervisors) Wageningen Agricultural University, The Netherlands (1998) 248 pp. ISBN 90-5485-874-5. PhD thesis. June 1998.

[19] Tom Heskes et al. *Causal Shapley Values: Exploiting Causal Knowledge to Explain Individual Predictions of Complex Models*. arXiv:2011.01625 [cs]. Nov. 2020. DOI: 10.48550/arXiv.2011.01625. URL: http://arxiv.org/abs/2011.01625 (visited on 24th Apr. 2023).

[20] Hui Huang, Jinniao Qiu and Konstantin Riedl. *On the Global Convergence of Particle Swarm Optimization Methods*. arXiv:2201.12460 [cs, math]. June 2022. DOI: 10.48550/arXiv.2201.12460. URL: http://arxiv.org/abs/2201.12460 (visited on 30th May 2023).

[21] *Introducing ChatGPT*. en-US. URL: https://openai.com/blog/chatgpt (visited on 24th Apr. 2023).

[22] R. Eberhart J. Kennedy. 'Particle swarm optimization'. In: (Aug. 2002). URL: https://ieeexplore.ieee.org/document/488968.

[23] Roy E. Lave. 'A Markov Decision Process for Economic Quality Control'. In: *IEEE Transactions on Systems Science and Cybernetics* 2.1 (1966), pp. 45–54. DOI: 10.1109/TSSC.1966.300078.

[24] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. arXiv:1509.02971 [cs, stat]. July 2019. DOI: 10.48550/arXiv.1509.02971. URL: http://arxiv.org/abs/1509.02971 (visited on 24th Apr. 2023).

[25] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. arXiv:1705.07874 [cs, stat]. Nov. 2017. DOI: 10.48550/arXiv.1705.07874. URL: http://arxiv.org/abs/1705.07874 (visited on 24th Apr. 2023).

[26] Mark Meerschaert. *Mathematical Modeling*. URL: https://www.sciencedirect.com/book/9780123869128/mathematical-modeling (visited on 28th Jan. 2013).

[27] Volodymyr Mnih et al. 'Human-level control through deep reinforcement learning'. en. In: *Nature* 518.7540 (Feb. 2015). Number: 7540 Publisher: Nature Publishing Group, pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://www.nature.com/articles/nature14236 (visited on 31st May 2023).

[28] Elisa Moisi. 'Particle Swarm Optimization and Genetic Algorithms'. In: *Journal of Computer Science and Control Systems* 2 (1st Oct. 2009).

[29] Christoph Molnar. *Interpretable Machine Learning*. URL: https://christophm.github.io/interpretable-ml-book/ (visited on 2nd May 2023).

[30] J. A. Nelder and R. Mead. 'A Simplex Method for Function Minimization'. In: *The Computer Journal* 7.4 (Jan. 1965), pp. 308–313. ISSN: 0010-4620. DOI: 10.1093/comjnl/7.4.308. URL: https://doi.org/10.1093/comjnl/7.4.308 (visited on 24th Apr. 2023).

[31] Martijn van Otterlo and Marco Wiering. 'Reinforcement Learning and Markov Decision Processes'. en. In: *Reinforcement Learning: State-of-the-Art*. Ed. by Marco Wiering and Martijn van Otterlo. Adaptation, Learning, and Optimization. Berlin, Heidelberg: Springer, 2012, pp. 3–42. ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_1. URL: https://doi.org/10.1007/978-3-642-27645-3_1 (visited on 31st May 2023).

[32] Judea Pearl. *A Probabilistic Calculus of Actions*. arXiv:1302.6835 [cs]. Feb. 2013. DOI: 10.48550/arXiv.1302.6835. URL: http://arxiv.org/abs/1302.6835 (visited on 1st May 2023).

[33] Judea Pearl and Dana Mackenzie. *The Book of Why: The New Science of Cause and Effect*. URL: https://www.amazon.com/Book-Why-Science-Cause-Effect/dp/046509760X (visited on 15th May 2018).

[34] *Randomised controlled trial: comparative studies*. en. Oct. 2021. URL: https://www.gov.uk/guidance/randomised-controlled-trial-comparative-studies (visited on 31st May 2023).

[35] David B. Resnik. 'Randomized Controlled Trials in Environmental Health Research: Ethical Issues'. en. In: *Journal of environmental health* 70.6 (2008). Publisher: NIH Public Access, p. 28. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2653276/ (visited on 31st May 2023).

[36] Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. arXiv:1602.04938 [cs, stat]. Aug. 2016. DOI: 10.48550/arXiv.1602.04938. URL: http://arxiv.org/abs/1602.04938 (visited on 24th Apr. 2023).

[37] Cedar Riener and Daniel Willingham. 'The Myth of Learning Styles'. In: *Change: The Magazine of Higher Learning* 42 (Aug. 2010), pp. 32–35. DOI: 10.1080/00091383.2010.503139.

[38] Nimish Santhosh. *LunarLander-Custom*. original-date: 2020-11-03T08:40:01Z. Nov. 2020. URL: https://github.com/nimishsantosh107/LunarLander-Custom (visited on 9th Mar. 2023).

[39] John Schulman et al. *Proximal Policy Optimization Algorithms*. arXiv:1707.06347 [cs]. Aug. 2017. DOI: 10.48550/arXiv.1707.06347. URL: http://arxiv.org/abs/1707.06347 (visited on 24th Apr. 2023).

[40] *scipy.optimize.basinhopping*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.basinhopping.html#scipy.optimize.basinhopping. Accessed: 2023-05-29.

[41] *Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations — Stable Baselines3 2.0.0a5 documentation*. URL: https://stable-baselines3.readthedocs.io/en/master/ (visited on 24th Apr. 2023).

[42] Rainer Storn and Kenneth Price. 'Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces'. en. In: *Journal of Global Optimization* 11.4 (Dec. 1997), pp. 341–359. ISSN: 1573-2916. DOI: 10.1023/A:1008202821328. URL: https://doi.org/10.1023/A:1008202821328 (visited on 24th Apr. 2023).

[43] Ronald L. Rivest Thomas H. Cormen Charles E. Leiserson. *Introduction to Algorithms, 3rd Edition*. URL: https://www.amazon.com/Introduction-Algorithms-3rd-MIT-Press/dp/0262033844 (visited on 31st July 2009).

[44] David J. Wales and Jonathan P. K. Doye. 'Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms'. In: *The Journal of Physical Chemistry A* 101.28 (July 1997). Publisher: American Chemical Society, pp. 5111–5116. ISSN: 1089-5639. DOI: 10.1021/jp970984n. URL: https://doi.org/10.1021/jp970984n (visited on 24th Apr. 2023).

[45] Eric W. Weisstein. *Global Optimization*. en. Text. Publisher: Wolfram Research, Inc. URL: https://mathworld.wolfram.com/ (visited on 30th May 2023).

[46] Y Xiang et al. 'Generalized simulated annealing algorithm and its application to the Thomson model'. en. In: *Physics Letters A* 233.3 (Aug. 1997), pp. 216–220. ISSN: 0375-9601. DOI: 10.1016/S0375-9601(97)00474-X. URL: https://www.sciencedirect.com/science/article/pii/S037596019700474X (visited on 24th Apr. 2023).

CHAPTER

Appendix A

A.1 Probability of a state being a possible return value of m_t

Our goal with this attachment is to explain more in-depth why the probability that an arbitrary state is one of m_t's possible return values is $\left(1 - \left(1 - \left|S_o \cap S^{\mathcal{S}}\right| / \left|S^{\mathcal{S}}\right|\right)^{|\mathcal{O}|}\right)$.

This is given that we know $\left|S_o \cap S^{\mathcal{S}}\right|$; the number of state nodes for one return value of m_t (return value S_o). Considering this is the only return size we know, we assume that the other return values have the same expected size. Therefore, we assume that a node from another set has a probability of being part of the explanation equal to the percentage of $\left|S_o \cap S^{\mathcal{S}}\right|$ in $\left|S^{\mathcal{S}}\right|$. More mathematically expressed: $\left|S_o \cap S^{\mathcal{S}}\right| / \left|S^{\mathcal{S}}\right|$.

Figure A.1 hopefully helps the reader to get a good understanding. $|\mathcal{O}| = 3$, meaning m_t returns three sets of nodes, all with an expected size $|S_o| = 3$. The expected sizes $\left|S_o \cap S^{\mathcal{S}}\right|$ and $\left|S_o \cap S^{\mathcal{A}}\right|$ are two and one, respectively. We are mainly interested in a state being a possible return, meaning $\left|S_o \cap S^{\mathcal{S}}\right| = 2$ is our main focus.

Since $\left|S_o \cap S^{\mathcal{S}}\right| / \left|S^{\mathcal{S}}\right|$ is the probability that a node is part of an arbitrary return set, $1 - \left|S_o \cap S^{\mathcal{S}}\right| / \left|S^{\mathcal{S}}\right|$ is the probability that it is not. The probability that the node is not in any of m_t's return value is, therefore $\left(1 - \left|S_o \cap S^{\mathcal{S}}\right| / \left|S^{\mathcal{S}}\right|\right)^{|\mathcal{O}|}$. Finally, the opposite of a node not being a part of any return values is that it is a part of one or more. The probability that an arbitrary state is one of m_t's possible return values is, therefore, $\left(1 - \left(1 - \left|S_o \cap S^{\mathcal{S}}\right| / \left|S^{\mathcal{S}}\right|\right)^{|\mathcal{O}|}\right)$.

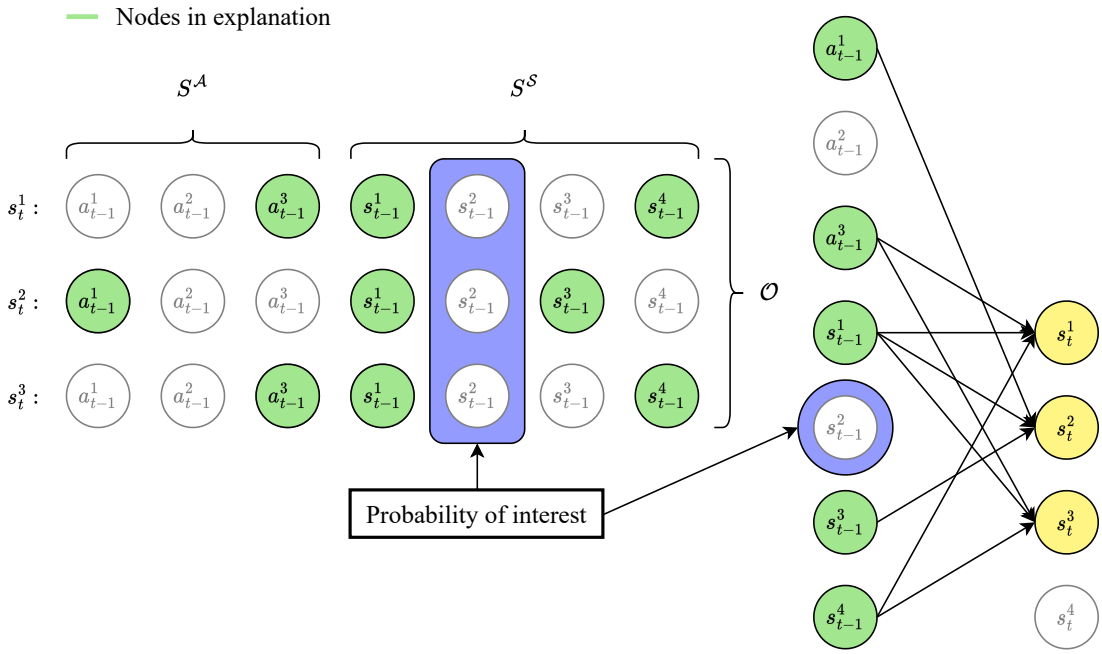
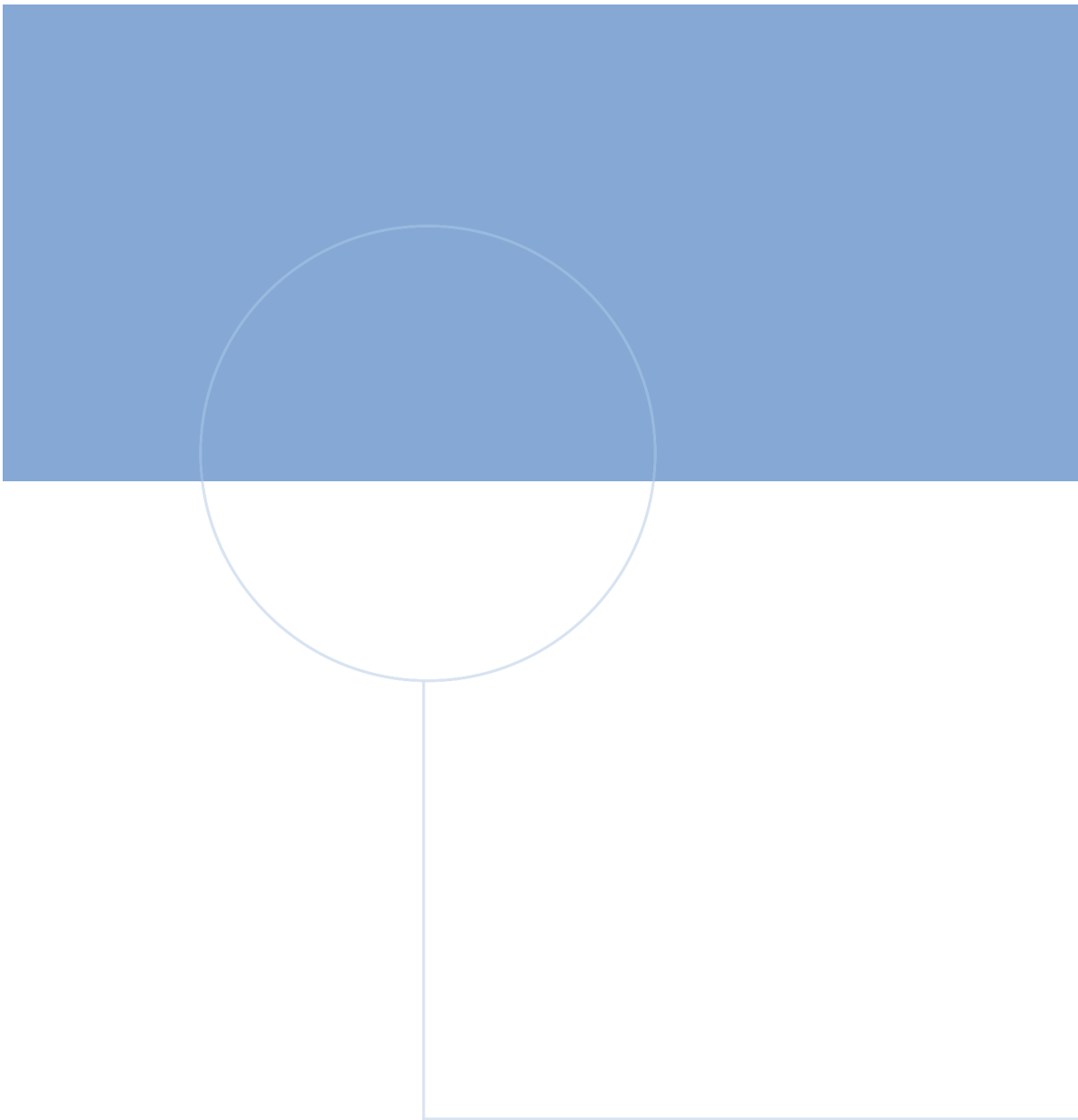


Figure A.1: Visualize the probability of a state being a possible return value of m_t



NTNU

Norwegian University of
Science and Technology