NTNU
Norwegian University of
Science and Technology

Explainable AI (XAI) methods for Deep Reinforcement Learning

Iver Osborg Myklebust

A project which precedes a Master of Science in Cybernetics and Robotics
Supervisor: Anastasios Lekkas, ITK
Submission date: December 2020

TTK4550 - Engineering Cybernetics, Specialization Project

Preface

This report was written based on my work on the pre-project of my master's thesis at NTNU, during the fall of 2020 under the supervision of Anastasios Lekkas.

The goal in this project is to develop and implement Explainable Artificial Intelligence (XAI) methods capable of extracting information from deep neural networks trained via reinforcement learning. Two of the most frequently used XAI methods today, Local Interpretable Model-Agnostic Explanation (LIME) and SHapley Additive exPlanations (SHAP), have been used in experiments across three different simulated robotic environments from OpenAI Gym and Robotis [1][2]. The implementation is done within an Anaconda environment in Jupyter Notebook, based on the machine learning library Pytorch. The Github libraries of SHAP, LIME and InterpretML are used to provide the interpretations [3][4][5]. The research has been implemented on a work station provided by NTNU.

The pre-project mainly serves as a learning period to prepare for the master's thesis, but the research has led to a good understanding of the XAI methods and how they work in robotic environments. I have also learned a lot about scientific writing and how to structure a project over a complete semester. The thesis got much more extensive than planned, but I wanted to write a report that do not require any background knowledge about XAI. Therefore, the methods and theory are thoroughly explained. However, it is assumed that the reader has a basic understanding of machine learning and mathematics. All figures have been created by the author with draw.io unless otherwise stated.

A huge thank you goes to Anastasios Lekkas for his encouraging support and supervision throughout the semester. Also a big thank you to PhD-student Sindre B. Remman for his insights and help with understanding the XAI libraries. His master's thesis, *Robotic manipulation using Deep Reinforcement Learning*, has served as a basis for some parts of this project [6]. The robotic manipulator is used as one of the experiments and the SHAP implementation is extended into more methods. Lastly, a special thank you goes to my friends and family in a tough semester characterized by Covid-19 restrictions and a demanding workload.

Abstract

The advancements in Artificial Intelligence (AI) the last decade has paved the way for an innovative and more digitalized society. To this date, AI solutions are present in many situations we encounter in our everyday life. Based on this evolution, more research is now being done into Reinforcement Learning (RL). For robotics, this could be a game changer, since RL makes systems able to learn from experience, which can take future robots into an even higher degree of autonomy. The biggest breaktrough is present in the field of Deep Reinforcement Learning (DRL), where RL is combined with Artificial Neural Networks (ANN).

However, the introduction of neural networks into reinforcement learning comes at a cost. The systems behave like a black-box, and rarely provide any explanations or justifications for their predictions. In robotics, where mistakes could lead to catastrophic consequences, these systems must be made more transparent and trustworthy. The aim for Explainable Artificial Intelligence (XAI) is to interpret an agent's decision-making to obtain insight into the black-box systems.

Two XAI methods have been used in this project, SHapley Additive exPlanations (SHAP) and Local Interpretable Model-Agnostic Explanation (LIME). These have been implemented in three robotic environments trained with DRL-algorithms. The environments used are *Cartpole-v1* and *LunarLander-v2* from OpenAI Gym and OpenMANIPULATOR-X by ROBOTIS. Local and global explanations that interpret trained agents have been explored across these three environments.

The results showed that XAI methods gave more insight into robotic DRL models, and many of the explanations were plausible. However, it also revealed that these methods have limitations in more complex environments, especially when features are highly correlated. This could be an indication that XAI methods should be used in a critical sense together with other tools to get more transparency. In the end, a comparison between the two XAI methods is given, based on their interpretations and properties.

This project is a part of the EXAIGON project [7], and will be continued in the spring of 2021 during the work on the author's master's thesis. SHAP, LIME and other XAI methods will be explored on relevant use cases provided by industry players.

Sammendrag

Fremskrittene i kunstig intelligens (AI) det siste tiåret har banet vei for et innovativt og mer digitalisert samfunn. Til dags dato er AI-løsninger til stede i mange situasjoner vi møter hver dag. Basert på denne evolusjonen blir det nå gjort mer forskning på forsterket læring (RL). For robotikk kan dette være en gamechanger, siden RL gjør systemer i stand til å lære basert på erfaring, noe som kan øke graden av autonomi i fremtidens roboter. Det største gjennombruddet er tilstede i feltet dyp forsterkende læring (DRL), der RL kombineres med kunstige nevrale nettverk (ANN).

Imidlertid kommer innføringen av nevrale nettverk i RL med en ulempe. Systemene oppfører seg som en svart boks, og gir sjelden noen forklaringer eller begrunnelser for deres prediksjoner. I robotikk, der feil kan føre til katastrofale konsekvenser, må disse systemene gjøres mer transparente og pålitelige. Målet for forklarende kunstig intelligens (XAI) er å tolke beslutningene til en agent for å få mer innsikt og forståelse i disse systemene.

To XAI-metoder har blitt brukt i dette prosjektet, SHapley Additive exPlanations (SHAP) og Local Interpretable Model-Agnostic Explanation (LIME). Disse er implementert i tre robotmiljøer trent med DRL-algoritmer. Miljøene som brukes er *Cartpole-v1* og *LunarLander-v2* fra OpenAI Gym og OpenMANIPULATOR-X av ROBOTIS. Lokale og globale forklaringer som tolker trente agenter har blitt utforsket i disse tre miljøene.

Resultatene viste at XAI-metoder ga mer innsikt i robotaktige DRL-modeller, og at mange av tolkningene var plausible. Imidlertid viste det også at disse metodene har begrensninger i mer komplekse miljøer, spesielt når funksjoner er sterkt korrelerte. Dette kan være en indikasjon på at disse metodene bør brukes i kritisk forstand sammen med andre verktøy for å få mer åpenhet. Til slutt blir det gitt en sammenligning mellom de to XAI metodene, basert på deres tolkninger og egenskaper.

Dette prosjektet er en del av EXAIGON-prosjektet [7], og vil fortsette våren 2021 under arbeidet med forfatterens masteroppgave. SHAP, LIME og andre XAI metoder vil bli utforsket over relevante problemstillinger fra bransjen.

Contents

Preface i

Abstract ii

Sammendrag iii

List of tables viii

List of figures xi

Acronyms xi

1 Introduction 1

1.1 Background and motivation 1

1.2 Objectives and research questions 6

1.3 Contributions 7

1.4 Outline of the report 8

2 Theory 9

2.1 Machine learning 9

2.2 Reinforcement learning 10

2.2.1 Value Functions 11

2.2.2 Exploration and exploitation 12

2.2.3 Q-learning 12

2.3 Deep Learning 13

2.3.1 Artificial Neural Networks . . . 14
2.3.2 Deep Reinforcement Learning . . . 15
2.4 Explainable Artificial Intelligence . . . 17
2.4.1 Surrogate models and local explainability . . . 17
2.5 LIME . . . 18
2.5.1 Lime Tabular . . . 19
2.6 SHAP . . . 20
2.6.1 Shapley Values . . . 20
2.6.2 DeepLIFT . . . 22
2.6.3 DeepExplain . . . 23
2.7 Morris sensitivity . . . 24
2.8 Partial Dependence Plots . . . 25

3 Methodology and experiments **27**
3.1 Software . . . 28
3.2 Environments . . . 29
3.3 Methodology . . . 33

4 Results **37**
4.1 Cartpole-v1 . . . 37
4.2 Lunar Lander . . . 46
4.3 Robotic Manipulator . . . 50

5 Discussion and analysis **55**
5.1 SHAP . . . 56
5.2 LIME . . . 59
5.3 XAI or XRL? . . . 59

6 Conclusion **61**
6.1 Answering the research questions . . . 61
6.2 Further work . . . 63

References **65**

List of Tables

3.1 State space Cartpole-v1 29
3.2 State space Lunar-Lander v2 30
3.3 Action space Lunar-Lander v2 31

List of Figures

1.1 DARPA's three waves of AI, from [8] . . . 1
1.2 The black box problem . . . 2
1.3 Panda example taken from [13] . . . 3

2.1 Reinforcement Learning process . . . 11
2.2 Artificial Neural Network example taken from [26] . . . 15
2.3 Model-Agnostic methods . . . 18
2.4 Lime function from [31] . . . 19
2.5 Explaining individual flu predictions with LIME [30] . . . 20
2.6 Shapley Values feature effects, from [34] . . . 22
2.7 Neural Networks consist of many simple components, from [32] . . . 24

3.1 Cartpole schematic drawing . . . 30
3.2 Lunar Lander schematic drawing from [43] . . . 31
3.3 Robotic Manipulator from [6] . . . 32
3.4 Example of a Decision plot . . . 34
3.5 Example of a Force plot . . . 35

4.1 Cartpole: Schematic figure for Example 1 with feature values . . . 38
4.2 Cartpole: LIME explanations for Example 1 . . . 38
4.3 Cartpole: SHAP Decision plot Example 1 . . . 39
4.4 Cartpole: SHAP Force plot for Example 1 . . . 39
4.5 Cartpole: Schematic figure for Example 2 . . . 40
4.6 Cartpole: LIME explanations for Example 2 . . . 41
4.7 Cartpole: SHAP Decision plot for Example 2 . . . 41

4.8 Cartpole: SHAP Force plot for Example 2 41
4.9 Cartpole: SHAP Global Summary plot over 5 episodes 42
4.10 Cartpole: Global Morris Sensitivity plot over 5 solved episodes 43
4.11 Cartpole: Partial dependence plot Pole Angle 44
4.12 Cartpole: Partial dependence plot Cart Velocity 44
4.13 Cartpole: SHAP Global Decision plot 45
4.14 Lunar Lander: Example scheme 46
4.15 Lunar Lander: LIME prediction probabilities 47
4.16 Lunar: LIME plot for action **Fire Main Engine** 47
4.17 Lunar Lander: SHAP Force plot for action **Fire Main Engine** 48
4.18 Lunar Lander: SHAP Local Summary plot 48
4.19 Lunar Lander: SHAP Global Summary plot 49
4.20 Robotic Manipulator: Situation 1 50
4.21 Robotic Manipulator: Shap Local Summary plot 51
4.22 Robotic Manipulator: LIME prediction probabilities 51
4.23 Robotic Manipulator: LIME explanations **Joint 1** 52
4.24 Robotic Manipulator: SHAP Force plot **Joint 1** 52
4.25 Robotic Manipulator: LIME explanations **Joint 3** 53
4.26 Robotic Manipulator: SHAP Force plot **Joint 3** 53
4.27 Robotic Manipulator: SHAP Global Summary plot 54

5.1 Cartpole: Correlation matrix over 5 episode 58
5.2 Lunar Lander: Correlation matrix over 5 episodes 58

Acronyms

AI Artificial Intelligence. ii, iii, ix, 1–3, 9, 13

ANN Artificial Neural Networks. ii, iii, 14–16

DDPG Deep Deterministic Policy Gradient. 31

DL Deep Learning. 2

DRL Deep Reinforcement Learning. ii, iii, 6–9, 14, 33, 37, 55, 60, 61, 63

GDPR General Data Protection Regulation. 3

LIME Local Interpretable Model-Agnostic Explanation. i–iii, 6–8, 17–20, 27, 31, 33, 35, 37, 39, 40, 47, 50, 51, 56, 57, 59, 62

MDP Markov Decision Process. 10–12

PDP Partial Dependence Plot. 25, 43

RL Reinforcement Learning. ii, iii, 1, 7, 10, 11, 14, 16, 28

SHAP SHapley Additive exPlanations. i–iii, 6–8, 17, 22, 23, 27, 31, 33, 34, 36–40, 42, 43, 45–51, 53–57, 59, 62

XAI Explainable Artificial Intelligence. i–iii, 2, 4, 6–9, 17, 25, 27, 31, 33, 55–57, 60–63

Chapter 1

Introduction

1.1 Background and motivation

Intelligent systems research has made huge progress in the last decade, and human-machine interaction is a part of our everyday-life. DARPA has split the progress in Artificial Intelligence in three waves (Fig 1.1) [8]. In the first wave, sets of logic rules were created to represent knowledge in limited domains. This enabled reasoning over narrowly defined problems, but the algorithms had no learning capability and didn't handle uncertainty well. With the progress in machine learning, statistical models are created for specific problem domains and trained on big data. In the second wave, there are three main types of learning, supervised and unsupervised which are driven by tasks and data, and Reinforcement Learning which aims to solve problems through experience and error handling [9].

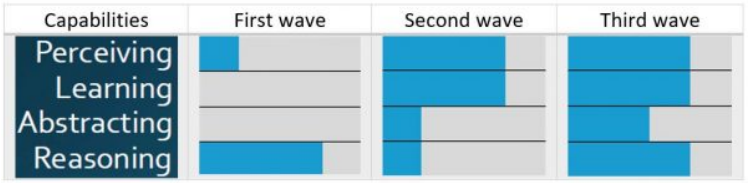

Figure 1.1: DARPA's three waves of AI, from [8]

However, great prediction capabilities also meant minimal reasoning ability. This is because the statistical models have many parameters that do not correspond to intuitive variables. It all leads to the development of a third wave, Explainable Artificial Intelligence, where statistical learning meets transparency and reasoning. The next section will start by describing some of the incentives behind the development of XAI.

Towards transparency

Artificial Intelligence has made tremendous progress recently with the introduction of Deep Learning (DL). By using statistical learning together with neural network classifiers, a huge number of practical applications have emerged. The effectiveness of the neural nets are achieved by passing the input through many hidden layers with millions of parameters and different activation functions [10]. In many ways neural networks are spreadsheets on steroids. By stretching and squashing each layer many times, a complex space can be represented in lower dimensional structures. This leads to systems that can do everything from facial recognition to beat humans in video games [8][9].

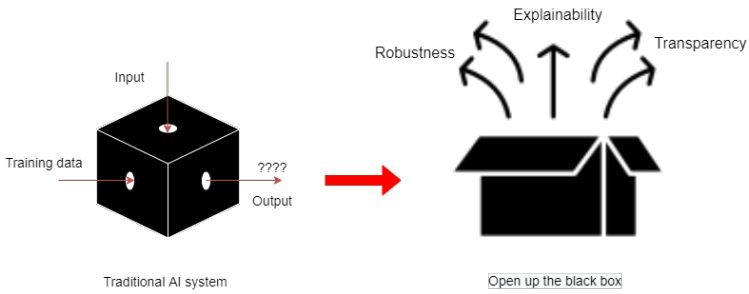

Figure 1.2: The black box problem

With the progress in machine learning, AI has become a widespread and powerful tool in our daily life. However, the problem with neural classifiers is the lack of transparency. They behave like a black box shown in Fig 1.2, where it is difficult to get insight into the mechanisms that produce the output [9]. In many situations, like life changing decisions in hospitals or on the roads, the need for trusting these critical systems is crucial.

A common example in the literature about transparent deep learning is the panda picture (Fig 1.3), where one of the most powerful neural classifiers correctly predicts a panda at first. However, when adding a small adversarial noise, it gets tricked to confidently predict a gibbon. This shows that even the best classifiers sometimes can be vulnerable to adversarial attacks, small perturbations in the input can bring big changes in the output. In critical systems this can lead to tragic mistakes, for example when an autonomous Uber car killed a woman [9][11][12]. With more transparency, these mistakes can be recognized in an earlier stage, and makes it easier for the human operator to fix the problem.

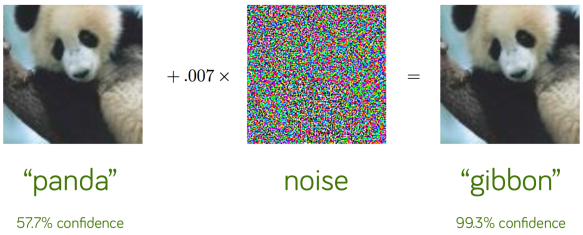

Figure 1.3: Panda example taken from [13]

The panda picture in Fig 1.3 is an example of adversarial AI. The consistent misclassification of adversarial examples is something many neural networks are struggling with. Such examples are inputs formed by applying small but intentional worst-case perturbations to the data. The cause of these mistakes was long a mystery, and some argued it happens because of overfitting and non-linearity. However, new research instead points at the problem with the models being too linear [13]. In the future, more powerful optimization methods can perhaps get rid of the adversarial mistakes, but adversarial training demands increased transparency.

The EU’s General Data Protection Regulation (GDPR) has recently added the right to explanation in some of its articles, leading the way for the transition towards transparent human-machine interaction. It is not clear how these legal requirements will be implemented in practice, but it will definitely increase the focus on trying to make such

systems in the years to come [14]. Legislations made by experts set the rules for the industry when the computers get even more powerful. It will hopefully also clarify and simplify some of the terms used in today's literature.

The ideal human-machine interaction

How do we explain our choices? And how do we want computers to explain the decisions made? These are difficult questions without clear answers, and dependent on the specific situation. There exist several goals and types of transparency. For a developer, it is important to understand how the system works, to be able to debug or improve it. From the user's perspective, it is more important to get a sense of trust in the technology, to get comfortable with the predictions [15]. This can help overcome the fear of the black box or be able to explain weird results. As mentioned in the last section, transparency is also important to facilitate safety testing and legal liability in critical systems.

It is important to separate between situations where technology is used as an extra tool and situations where technology is used in alternative to a person. For example in an emergency room, XAI can be used to help the doctor take the correct decisions while a bank system might need to explain its final decision of why the customer's loan application got rejected. The amount of transparency is probably the most important design decision for the developer when making explainable systems. When it comes to cyber-physical systems, in addition to inspiring trust, transparency and explainability can help a lot when something has gone wrong. It makes it possible to go to the logged data, investigate what went wrong and engineer a better solution that will avoid a similar phenomena in the future.

Lipton [16] defines transparency as the opposite of a black-box-ness. Transparency gives an understanding of how the model works. The amount of transparency needed is dependent of the specific system, and today it has often been a trade-off between transparency and prediction accuracy. The goal is to keep the high prediction rate of deep learning and make these systems more transparent. However, full transparency may not always be possible or fully required. Therefore, when defining the background theory behind Explainable Artificial Intelligence, interpretability will be used. Interpretability

refers to which extent a learned model makes sense to a user while explanations is a way to clarify how the learned model works. To better understand how to extract information from deep neural networks trained via reinforcement learning, more experiments and research is needed. The motivation of this thesis is therefore to investigate how explainable methods can improve the understanding and make these systems more trustworthy.

1.2 Objectives and research questions

The main goal of this thesis is to answer the following research questions:

- **Can methods from Explainable Artificial Intelligence (XAI) be used to interpret the results on models trained with Deep Reinforcement Learning (DRL)?**
- **Of the two most common XAI methods today, LIME and SHAP, which framework is best suited to robotic applications in real-life environments? What are the important factors to consider when choosing between the two methods?**

To answer these questions, the semester was divided into a series of objectives to track the project progress:

1. Literature review of XAI
2. Get familiar with the most common XAI methods, decide which methods to use in the project.
3. Become familiar with, and build on, the MSc thesis of Sindre B. Remman, who developed SHAP explanations on a robotic manipulator. Expand the explanations and implement LIME. Multiple challenges with both methods were discovered. Decided to explore in simpler environments with fewer states and actions.
4. OpenAI Gym was used to do attempts on simpler environments with pre-solved deep learning algorithms. Implemented SHAP and LIME on *Cartpole-v1* and *LunarLander-v2*.
5. Compare local SHAP and LIME explanations. Force and decision plots (Chapter 3) were implemented from the SHAP library, together with changing into prediction probabilities, to make the interpretations easier to compare.
6. Use the the open-source package InterpretML that incorporates state-of-the-art machine learning interpretability techniques, to compare global SHAP results with more traditional methods [5]. Investigate further on the advantages and disadvantages of each method and how this can be used in future master's thesis work.

1.3 Contributions

- Three pre-trained reinforcement learning environments, described in Chapter 3, are used to show how some of the most promising XAI-methods can interpret the decisions of a DRL agent. The environments used are from the control package of OpenAI gym and a robotic manipulator trained by Sindre B. Remman in his master's thesis work in the spring 2020.
- The package of SHAP are used to produce both local and global interpretations across all environments. The LIME package has been used to produce local interpretations in the same environments, and gets compared to the SHAP predictions in Chapter 4. In addition, to better understand the model's global behavior, Interpret ML has been used to compare the global SHAP interpretations with traditional feature importance techniques.
- This project combines two state of the art techniques that still are mostly in the development phase. DRL for robotic problems is getting increased attention and XAI methods have started to be explored in supervised machine learning problems. The thesis shows that XAI methods can give some interpretations in simple environments that are on-line with human intuition. Such insight can increase or decrease the trust of using these DRL-models in real-life robotic problems. However, it struggles more when the dimensionality of the environments increases with highly correlated features.
- In this thesis, it leads to a discussion in Chapter 5 about the assumptions being made in these methods, and how this can be deployed when trying to interpret RL-agents with no clear answers. The conclusion is to keep the human side of the equation in mind. XAI methods can be used as a tool to interpret results in DRL environments, but the limitations of the methods in correlated environments should put the explanations under a critical view.

1.4 Outline of the report

The rest of the report is divided into six chapters:

- Chapter 2: **Theory**
  - This chapter introduces terminology and theory that is important for the rest of the thesis. It starts with an overview of the DRL methods used to train the models before explaining the theoretical foundation behind the XAI methods (LIME and SHAP). In the end, possible methods to understand global feature values, Morris sensitivity and Partial dependence plot, are introduced.

- Chapter 3: **Methodology and experiments**
  - An overview of the main software used in this thesis is provided. The three environments are presented together with their implementations. The function and features from the XAI-packages will also be explained to give a better understanding of the results in the next chapter.

- Chapter 4: **Results**
  - Local and global results from all three environments are presented. The results are discussed briefly in comparison to what the author would have interpreted and explained in the same situations.

- Chapter 5: **Discussion and analysis**
  - The results from Chapter 4 are discussed further. This will be analyzed in context with properties of the XAI methods.

- Chapter 6: **Conclusion**
  - A conclusion to the thesis is given. The research questions are answered and in the end the possible extensions into the author's master's thesis (future work) are described.

Chapter 2

Theory

This chapter will introduce the background theory used in this thesis. Since the project implements XAI methods on models trained with deep reinforcement learning, the theory behind DRL-algorithms will first be described. In contrast to traditionally more common control methods, reinforcement learning makes the robots able to learn from experience. This helps tackle uncertainty, but today it comes at the cost of less knowledge about the robots decisions. This chapter will also explain the theory behind the explainable methods that are used to derive more interpretations from the neural networks.

2.1 Machine learning

Oxford dictionary defines Artificial Intelligence as *"the theory and development of computer systems able to perform tasks normally requiring human intelligence"* [17]. Machine learning is a branch of AI which allows models to improve performance based on processed data. In other words, algorithms learn from experience. There are three main types of machine learning [18][19]:

- Supervised learning - The model learns by using labeled output data as guidance.
- Unsupervised learning - The model learns by finding patterns in unlabeled data without any guidance.

- Reinforcement learning - The model learns by interacting with the environment. In many ways more similar to how humans learn. An agent learns how to behave in an environment, and the goal is to maximize the feedback reward signal in the long run.

2.2 Reinforcement learning

A lot of literature exists on reinforcement learning, and depending on the problem different approaches can be relevant. This thesis will mostly focus on the literature behind the algorithms used to solve the environments used in this project. Similar to how the human brain learns, RL employs positive and negative feedback to learn how to perform various tasks. This can be compared with closed-loop problems where the goal is to maximize the reward. The environment defines the task that is going to be solved. It is modeled as a Markov Decision Process (MDP), and RL is a way to solve problems described by MDPs. An agent observes and acts on the environment based on the goal to maximize reward. To control the agent a decision process needs to be defined, inspired by [20] and [21].

- States - **S** is a set of states called the state space. A state is a unique characterization of all that is important in a state of the problem that is modelled.
- Actions - **A** is a set of actions called the action space. The set of actions that can be applied in some particular states $s \in S$ is denoted $A(s)$, and can be used to control the system state.
- Transition Function - The transition from a state s to a new state s', by applying action $a \in A$ to $s \in S$. This is based on a probability distribution over the set of possible transitions, and the function is defined as $T : SxAxS \rightarrow [0, 1]$.
- Reward Function - Used to give direction in which way the system (MDP) should be controlled. Defined as $R : SxAxS \rightarrow \mathbb{R}$, that implicitly specifies the goal of learning.
- Markov Decision Process - The four definitions above leads to definiton of an MDP, which is a tuple $\langle S, A, T, R \rangle$. The transition function and the reward function together define the MDP model. MDPs have three different optimality criteria.

  - Finite horizon $E[\sum_{t=0}^{h} r_t]$
  - Discounted, infinite horizon $E[\sum_{t=0}^{h} \gamma^t r_t]$, where $\gamma \rightarrow [0, 1]$ decides the trade-off between shortsighted and equally weighted approach.
  - Average reward $\lim_{h\rightarrow\infty} E[\frac{1}{h} \sum_{t=0}^{h} r_t]$

- Policies - Determines which action an agent should take based on the environment state. Can be deterministic (direct mapping) or stochastic (maps over a probability distribution). Defined by $\pi : S \rightarrow A$.

- RL Process - In a basic RL operation the agent receives a state from the environment, performs an action and receives a reward continuously. This is illustrated in Fig 2.1.

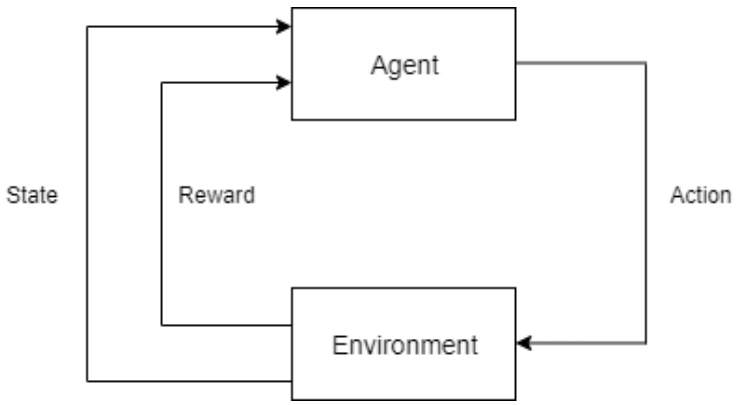

Figure 2.1: Reinforcement Learning process

2.2.1 Value Functions

Value functions link the optimality criteria to policies. The most common approach for the MDPs is to learn value functions to compute optimal policies. The value function is defined as the value of state s under policy π and denoted $V^{\pi}(s)$. An optimal policy, denoted π^*, is given by

$$V^{\pi^*}(s) \geq V^{\pi}(s) \qquad \forall s \in S \tag{2.1}$$

Model-based solutions are techniques where the probability or/and transition matrix are known. The two most common methods are value iteration and policy iteration. Policy iteration focuses on evaluating and improving the policy at every step, while value iteration focuses purely on estimating the value function and after it has converged towards V^* the policy is computed.

However, these two techniques assume that a perfect model is available. In reinforcement learning, such a model is often not defined since the probability and transition matrices are unknown. This is called model-free solutions, where statistical knowledge about the model is gathered through MDP sampling. One way to do this is called temporal difference learning, which uses a bootstrapping technique. This means it learns estimates of values based on other estimates, and was used when training these environments [6][20].

2.2.2 Exploration and exploitation

To define Q-learning - one of the most commonly used model-free algorithms in reinforcement learning, it is crucial to understand the trade-off between exploration and exploitation. This is what decides the agent's ability to discover new strategies, balancing between *exploiting the best actions* and *exploring the environment* by trying new strategies [18]. The most common exploration strategies are

- ϵ-greedy exploration - The agent choose the greedy (exploiting the best action) with probability ϵ, or a random action with probability $1 - \epsilon$.
- Exploration noise - To get the agent to discover new strategies along with using the learned knowledge from the environment, noise can be added to the greedy action. This demands a continuous action space, which is usually present in robotic problems.

2.2.3 Q-learning

Q-learning is a temporal difference learning algorithm. Instead of determining the state values from the value function in section 2.2.1, Q-learning estimates the quality of an action that is taken to move to a state. It is a tabular method, which means that the

learned Q-values are inserted into a state- and action space table, denoted by $|S|x|A|$. The update rule for the Q-learning algorithm is given by

$$Q(s,a) := Q(s,a) + \alpha\left(r + \gamma \max_{a' \in A(s')} Q(s',a') - Q(s,a)\right) \tag{2.2}$$

where α is the learning rate, r the reward and γ the discount factor. The algorithm is exploration insensitive under the assumption that a state-action can be visited infinite times so α can be decreased. This means it will converge to the optimal policy while following some exploration policy π [20]. A pseudocode of the algorithm is presented below.

Q-learning (off-policy TD control) taken from [21]

Require: discount factor γ, learning parameter α,
initialize Q arbitrarily (e.g. $Q(s,a) = 0, \forall s \in S, \forall a \in A$)
foreach *episode* **do**
  s is initialized as the starting state
  repeat:
  choose an action $a \in A(s)$ based on Q and an exploration strategy
  perform action a
  observe the new state s' and received reward r
  $Q(S,A) := Q(S,A) + \alpha[r + \gamma \max_{a' \in A(s')} Q(s',a') - Q(s,a)]$
  $s := s'$
  until s' is a goal state
end foreach

2.3 Deep Learning

Machine learning is a branch of Artificial Intelligence that provides systems the ability to learn from experience. Deep learning is a branch of machine learning that uses neural networks to solve complex problems. By using higher-level learned features defined in terms of lower-level features, deep learning seeks to exploit unknown structures in the input distribution to discover good representations [22]. One of the leaders of the Google Brain Project, Andrew Ng, has compared deep learning with a rocket engine.

With enormous amounts of fuel (data), the rocket needs powerful engines (deep learning models) to lift off the ground [23].

Before introducing the neural network that is used in deep reinforcement learning, some terms used in this section will be defined [24].

- **Perceptron** - A neuron in the human brain is a cell that transmits and processes information. Perceptrons are in many ways simplified versions of human brain cells that take several inputs and weigh them up to produce a single output. This was the first type of an artificial neuron, but it is not so commonly used today.
- **Activation function** - To calculate the weighted neuron, an activation function is used. It calculates a weighted sum of inputs, adds bias and from this information it decides what should be fired to the next neuron. The most popular functions used today are Tanh, Softmax and ReLU.
- **Gradient descent** - An algorithm to find the local minimum of a function. By guiding the solution in the direction of the steepest descent, it can be used to update the parameters of the model.
- **Back-propagation** - Algorithm used to calculate the gradient descent. The goal is to minimize the error between the input and output, and back-propagation is used to train the neural network to an acceptable error margin.

2.3.1 Artificial Neural Networks

In the last decade multiple types of neural networks have emerged, with different application features. For example, Convolutional Neural Networks (CNN) are often used in image processing, while Recurrent Neural Networks (RNN) have great ability in speech recognition. They are all a part of the broader family of Artificial Neural Networks (ANN). In this section ANNs will be defined, before the next section will introduce Deep Reinforcement Learning (DRL) by combining RL with the modelling power of ANNs.

ANNs consist of artificial neurons, and have five main components: inputs x, outputs y, weights w, biases b, and an activation function $f(...)$. The relationship between these

are given by

$$y = f(w^T x + b) \tag{2.3}$$

If this relationship gives 0 or 1 depending on the sign of the neural network it is called a perceptron. However, since this means a small change in the function's weights and biases can radically change the output, it is not commonly used. Other activation functions where small changes lead to a small output change are preferred.

When putting the artificial neurons into a network, this network is able to compute complex functions. Every ANN has at least one input and output layer, and can also have multiple hidden layers. Hyperparameters are parameters where the value is set before the learning starts, and the number of neurons in each layer (width) and the number of layers (depth) are such hyperparameters. The main advantage with ANNs is the ability to reproduce and model nonlinear processes [25]. An example is shown in Fig 2.2.

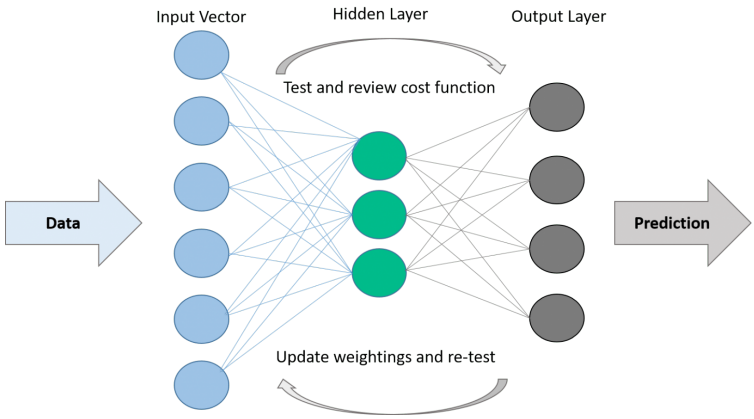

Figure 2.2: Artificial Neural Network example taken from [26]

2.3.2 Deep Reinforcement Learning

Robotic systems often have multiple joints, which means a high-dimensional state space. This is a challenge in reinforcement learning, because the volume of the space increases so fast, hence the available data become sparse. The problem is called the curse of

dimensionality, and it would take an enormous amount of computations, memory and time to explore.

However, by combining RL with ANNs, a successful approach to this problem was discovered. The idea is to use a nonlinear function approximator to map state and action to a value. A Deep-Q-Network was first created by DeepMind in 2013 [27]. Deep-Q-Network is based on the Q-learning algorithm in 2.2.3, and was able to learn control policies directly from a high-dimensional sensory input by using neural networks. To stabilize the training, a replay buffer and a target network were used [27].

- Replay buffer - By storing every transition between the explored samples in a replay buffer, and sample a minibatch randomly in every update, the samples are independent of each other. This improves the generalization of the neural network.
- Target network - Instead of using a target that changes with every timestep, a network is used to minimize the target correlation. This makes the training easier.

Three DRL-algorithms are used to train the different environments in this thesis. The trained model of each environment are used when implementing XAI-methods.

- REINFORCE (Monte Carlo Policy Gradient) - Learns a parameterized policy that can select actions without consulting a value function. Maximize performance by updating approximate gradient ascent in a stochastic estimate. Used to solve the **Cartpole**-environment.
- Deep-Q-Network - See the explanation above. Used in the **Lunar Lander**-environment.
- Deep-Deterministic Policy Gradient (DDPG) - Based on Deep-Q-Network, but in addition to learning a Q-function, it also learns a policy by using two target networks. Can be used in MDPs with continuous action spaces, and in contrast to DQN who work best with low-dimensional action spaces, DDPG can be used in higher-dimensional action spaces. This algorithm is used to solve the reacher task on the **Robotic Manipulator**.

2.4 Explainable Artificial Intelligence

The introduction of deep learning allows problems that seemed impossible before to be solved. However, such complex machine learning models have been treated as black boxes until now. Human interpretability of intelligent machines often leads to a trade-off with the model performance. When using simple linear regression models, the number of parameters makes it possible to explain the decisions made. With neural networks, this is much harder because of the enormous amount of connections. Even a narrow and shallow network can have tens of thousands connections [6]. The performance and accuracy of deep learning makes it tempting to use, but in many situations the client demands a more transparent system.

2.4.1 Surrogate models and local explainability

The idea behind Explainable Artificial Intelligence is to open up the black box by using some of the tools from simpler regression models. In such models, *Beta* coefficients are used to explain the prediction for all data points. This is called global fidelity, when a variable value increases by 1, prediction increases by *Beta* for every data point. But this does not explain the effect of individual data points, in other words why the effect from one user's variable change could be different from another. This is called local fidelity, and local function explanations often have the property of linear and monotonic local regions. LIME and SHAP provide interpretability to black-box models by explore and use the property of local explainability. This is used to build surrogate models, which tweaks the input slightly and test how the prediction changes [28]. If the model prediction change much by tweaking a variable value, that variable for that particular data point may be an important predictor and vice versa.

Surrogate models still treat the system as a black box, which is called model agnostic methods since they are separating the explanations from the model (Fig 2.3) [29]. By exploring these local regions the black box can be opened, and the difference between LIME and SHAP is how they build these surrogate models to make an interpretable data representation. The next two sections will take a deeper look into both methods.

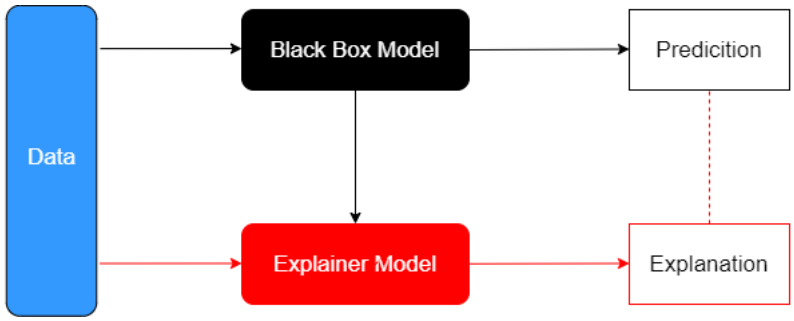

Figure 2.3: Model-Agnostic methods

2.5 LIME

The research paper *"Why Should I Trust You"* by Marco Ribeiro et al was one of the first to propose a technique to explain the black boxes of machine learning. It defines local Local Interpretable Model-Agnostic Explanation (LIME) as *an algorithm that can explain the predictions of any classifier or regressor in a faithful way, by approximating it locally with an interpretable model* [30].

Explaining a prediction means presenting artifacts, textual or visual, that improves the understanding between features (words, pixels or robotic joints) and the model's prediction. Interpretable explanations need to present interpretations that are understandable by humans. While the classifier may represent more complex features, LIME uses a binary vector $x' \in \{0,1\}^{d'}$ to represent an instance, where $x \in \mathbb{R}^d$. The objective is to minimize the difference in prediction response between the instance x and its neighbor.

An explanation model $g \in G$ is defined as a class of potentially interpretable models, for example linear models (Fig 2.4) or decision trees, with the domain $g \in \{0,1\}^{d'}$. $\Omega(g)$ is the measure of complexity of the explanation $g \in G$ as not every model is simple alone to be interpretable. Examples of the complexity, $\Omega(g)$, can be the depth of the three or the number of non-zero weights.

The model being explained is denoted $f : \mathbb{R}^d \rightarrow \mathbb{R}$, and $f(x)$ is the probability that

x belongs to a certain class. To define locality around x, $\pi_x(z)$ is defined as a proximity measure between an instance z to x. The fidelity function $L(f, g, \pi_x)$ is a measurement of how unfaithful g is in approximating f in the locality defined by π_x. This means L must be minimized while $\Omega(g)$ must be low enough to be interpretable by humans. The LIME explanation is defined by

$$\xi(x) = arg \min_{g \in G} L(f, g, \pi_x) + \Omega(g) \tag{2.4}$$

and this formulation can be used for multiple explanation models G, fidelity functions L and complexities Ω [30].

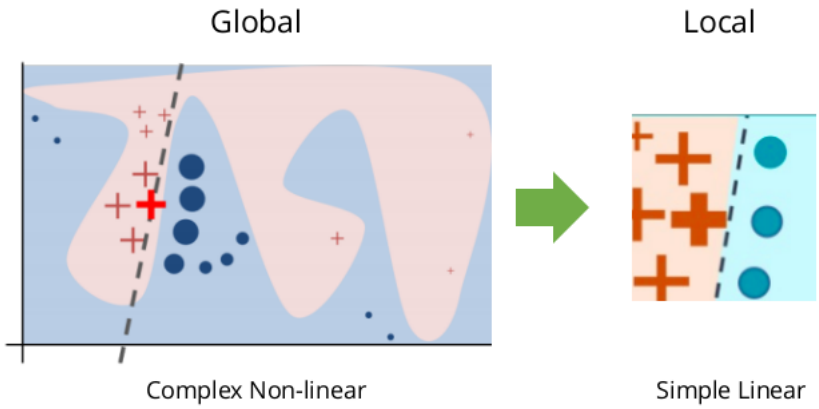

Figure 2.4: LIME localises a complex problem and defines a simpler linear model to explain a local prediction. Image taken from [31]

2.5.1 Lime Tabular

The LIME package is made for different datatypes. In this thesis, the environments generate matrix data so the tabular method is used. The function explains predictions on numerical features from the training data. This is done by perturbing them, hence sampling from a normal $(0, 1)$, and doing the inverse operation of mean-centering and

scaling. From this, neighborhood data is generated by randomly perturbing features, and locally weighted linear model can be generated from a learned classifier. The models can be used to explain each of the classes in an interpretable way.

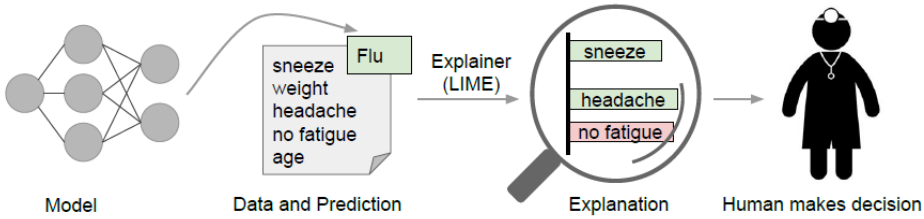

Figure 2.5: Explaining individual flu predictions with LIME [30]

2.6 SHAP

In the paper *A unified approach to interpreting model predictions* Scott M. Lundberg et al proposed a new explanation method based on Shapley additive feature attribution methods. This is also an approximation of the neural network, when the model is to complex to be understood by humans. The algorithm used in this thesis is called DeepSHAP. This algorithm is based on two foundations, the DeepLift algorithm and Shapley values, which will be presented first.

2.6.1 Shapley Values

As with LIME, the original prediction model is denoted by f and the explanation model by g. Simplified inputs, denoted by x', map to the original inputs using a mapping function $x = h_x(x')$. The goal of local methods is to make $g(z') \approx f(h_x(z'))$ given that $z' \approx x'$. This means an additive feature attribution can be defined

$$g(z') = \phi_0 + \sum_{t=1}^{M} \phi_t z'_t \tag{2.5}$$

where z' is a vector of binary variables with size M, M is the number of simplified input features and $\phi_t \in \mathbb{R}$ is an effect that is assigned to each feature [32].

The method of finding the effect values ϕ is based on game theory. The Shapley value is a solution concept in traditional cooperative game theory, to make a game "fair" according to the founder Lloyd Shapley [33]. To generate a total surplus of all players, four conditions must be met:

- The total reward should equal the sum of what everyone receives.
- The same amount of reward should be received from two people contributing the same value.
- No value contribution means nothing received.
- When playing two games, the individual's reward from both games should equal the reward sum from both the first and second game.

This can be transferred into three properties in Shapley value estimation:

- **Property 1 (Local accuracy)**

$$f(x) = g(x') = \phi_0 + \sum_{t=1}^{M} \phi_i x'_t \tag{2.6}$$

  This means the explanation model $g(x')$ matches the original model $f(x)$ when $x = h_x(x')$

- **Property 2 (Missingness)**

$$x'_t = 0 \rightarrow \phi_t = 0 \tag{2.7}$$

  Features missing in the original input have no impact.

- **Property 3 (Consistency)**
  If $f'(h_x(z')) - f'(h_x(z'_t = 0)) \geq f(h_x(z')) - f(h_x(z'_t = 0), \forall z' \{0,1\}^M$, then $\phi_t(f', x) \geq \phi_t(f, x)$.
  This means that if a model changes so a simplified input's contribution increases, that input's attribution should not decrease.

Only one explanation model g follows definition 2.5 and the three properties above:

$$\phi_t(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f(h_x(z')) - f(h_x(z'_t = 0))] \tag{2.8}$$

where $|z'|$ is the number of non-zero entries in z' and $z' \subseteq x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries in x'. Equation 2.8 gives the solution of the SHAP values, where each value indicates how much a given state contributes to the magnitude of a given output (Fig 2.6) [32].

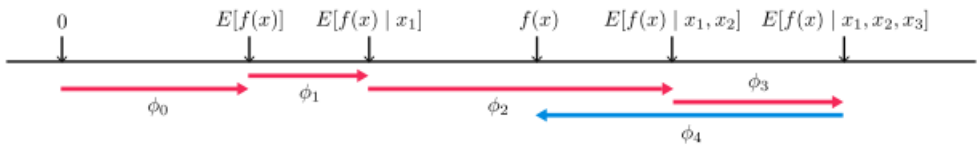

Figure 2.6: Shapley Values explain the output of a function f as a sum of the effects ϕ of each feature. Image taken from [34]

2.6.2 DeepLIFT

DeepLIFT is a recursive prediction explanation method for deep learning. With some modifications, this is an additive feature attribution method that converts binary values into the original inputs, $x = h_x(x')$. This binary value is decided by the effect of setting the input to the original value or a reference value that is decided by the user. This is denoted by $C_{\Delta x_t \Delta y}$, when $y = f(x)$ is the model output. DeepLIFT uses this in a "summation-to-delta" property

$$\sum_{t=1}^{n} C_{\Delta x_t \Delta y} = \Delta y \tag{2.9}$$

where $\Delta y = f(x) - f(r)$, $\Delta x_t = x_t - r_t$ and r the reference input. This matches equation 2.5 if we let $\phi_t = C_{\Delta x_t \Delta y}$ and $\phi_0 = f(r)$.

2.6.3 DeepExplain

It is challenging to compute exact SHAP values, but they can be approximated by combining insights from additive feature attribution methods. By assuming model linearity, the mapping can be approximated:

$$f(h_x(z')) \approx f([z_S, E[z_{\hat{z}}]] \tag{2.10}$$

where S is the set of non-zero indexes in z' and E the base value.

DeepExplain uses the connection between the DeepLIFT algorithm and the linear model approximation of Shapley values. By combining the two equations, and let the reference value from equation 2.9 represent $E[x]$ in equation 2.10, DeepLIFT approximates SHAP values.

This is equivalent to linearize the non-linear components of a neural network, through back-propagation rules for each component. Since DeepLift can be modified as an additive feature attribution method it satisfies the properties of Shapley values, and motivates adapting a method to approximate SHAP values for whole networks.

This method is called DeepExplain, and combines SHAP values computed for smaller network components by recursively passing DeepLIFT's multipliers backwards through the network (the composition rule). Fig 2.7 shows a simple component of a neural network, where the deepLIFT approximation is given by

$$m_{x_j f_3} = \frac{\phi_i(f_3, x)}{x_j - E[x_j]} \tag{2.11}$$

$$m_{y_i f_j} = \frac{\phi_i(f_j), y}{y_i - E[y_i]} \qquad \forall_j \in \{1, 2\} \tag{2.12}$$

$$m_{y_i f_3} = \sum_{j=1}^{2} m_{y_i f_j} m_{x_j f_3} \qquad \text{chain rule} \tag{2.13}$$

$$\phi_i(f_3, y) \approx m_{y_i f_3}(y_i - E[y_i]) \qquad \text{linear approximation} \tag{2.14}$$

Since such simple networks components can be solved efficiently if they are linear and a deep neural network consists of many simple components, the composition rule

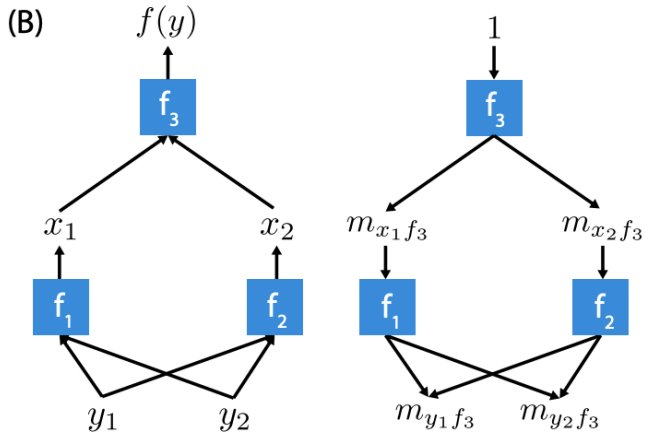

Figure 2.7: Neural Networks consist of many simple components, from [32]

enables a fast approximation for the full model. This makes it possible to explain deep neural networks efficiently [32].

2.7 Morris sensitivity

Sensitivity analysis corresponds to more traditional methods of capturing how a model reacts to change in inputs. It is defined as a measurement of describing relative importance of each input in determining output variability. This is very connected with covariance and correlation analysis, and can be done at a global or local level. One of the most common global methods is the Morris method [35].

The Morris method measures the sensitivity, defined as mu_{ij}, which are interpreted as the average influence of the input **i** on the model output **j**. This means that it changes *one-factor-at-a-time* to grasp the relative influence of each factor. In other words, it captures output variation when one of the inputs are moved to neighborhood points. The sensitivity analysis is similar to some of the building blocks the XAI frameworks are based on, but it is a simpler and a more traditional measurement. Morris method does

not split the feature predictions into different actions, it studies how the uncertainty in the output of a system can be divided into different sources of uncertainty in its inputs [36]. This can be helpful when testing the robustness of the model or understanding the relationships between the input and output variables.

2.8 Partial Dependence Plots

Partial Dependence Plot (PDP) shows the marginal effect one or two features have on the predicted outcome of a machine learning model [37]. It can show if a relationship between the feature and target is linear, monotonic or more complex, with the features x divided into two sets (S and C). x_S are the partial dependence features where the prediction effect will be plotted, while x_c are the other features used in the machine learning model. By marginalizing the machine learning model output over the x_c distribution, the function shows the relationship between the x_s features and the output, and a function that depends only on features in S can be defined.

$$\hat{f}_{xs}(x_S) = \frac{1}{n}\sum_{i_1}^{n}\hat{f}(x_s, x_c^{(i)}) \tag{2.15}$$

This partial function tells what the average marginal effect on the prediction is for given values of features in S. It is a global method that can be used when interpreting the global XAI-methods, and gives an understanding about the global relationship of a feature with the predicted outcome. However, PDP assumes independence between the features, and no correlation in robotic problems are unlikely. This could mean averages calculated does not reflect the true behaviour of the feature [38].

Chapter 3

Methodology and experiments

In this thesis, three models have been used to explore Explainable Artificial Intelligence on robotic systems. Both LIME and SHAP are relatively recent methods that mainly have been used to explore datasets and images. Therefore, the environments differ in state space, degree of explain difficulty and the amount of intuitive interpretation. In this chapter these experiments will be presented together with the software used in the implementation. Two of the environments are pre-trained deep learning models from the OpenAI Gym framework, while the last one is a continuation of Sindre Remman's master's thesis [6], where a robotic joint manipulator is trained with reinforcement learning.

The goal is to use these models to compare the explainable methods, and find possible indications of the approximations and assumptions. The results will be presented in the next chapter, and this will lead to a discussion about the interpretations and properties of the methods. As written in the introduction, the need to trust these critical systems are crucial when implementing them in the industry, and must be considered when using reinforcement learning instead of traditional control theory. Explainability can be an effective tool to increase the trust in black-box systems, but it is evenly important to be able to trust the given explanations.

In the end of this chapter an overview of how the XAI methods were implemented

will be presented. The functions and different plot types will be reviewed to give an easier interpretation of the results in the next chapter.

3.1 Software

PyTorch

PyTorch is a Python library for deep learning. It supports multiple features such as GPU for parallel computing, intuitive setups for common neural network techniques and effective debugging procedures. PyTorch's ability to support dynamic computation graphs makes it convenient and flexible to use compared to other deep learning frameworks such as Tensorflow, which uses static computation graphs [39].

Anaconda

Anaconda is an open-source distribution of Python for scientific computing that simplifies package management and distribution [40]. Anaconda is used together with Jupyter Notebook, an open-source web application that contains live code and visualizations. This is useful when dealing with multiple models and libraries where dependencies can arise, and it also makes it easier to explore explainability in pre-trained models.

OpenAI Gym

OpenAI Gym is an open-source toolkit for developing and comparing reinforcement learning algorithms. It aims to provide an easy to set up and standardized environment, so that published research becomes more easily reprocudible [1][41]. This also means that RL-algorithms can easily be adapted between environments. The environments used in this thesis are *LunarLander-v2* from the Box2D package and *CartPole-v1* from the classic control package. Box2D consists of continuous control tasks in 2D-simulators, while classic control consists of classic RL literature control theory problems. The main functions that need to be defined for Gym environments are [1]:

- make(environment_name) - Sets up a new instance of the environment and returns an object of the class.

- step(action) - Applies a step to the environment on the action used as the argument. Returns an observation of the environment, the transition reward, possible terminal state and a dictionary of diagnostic information specific to the environment.
- reset() - Resets and than returns the first observation of the reset environment.

3.2 Environments

Cartpole-v1

The *Cartpole-v1* environment is similar to an inverted pendulum with gravity center above its pivot point. The pole in Fig 3.1 is attached by an un-actuated joint to a cart, which moves along a friction-less track. The goal is to prevent the pendulum from falling over, which is defined as more than 15 degrees from the starting upright position. It is also not allowed to move the cart more than 2.4 units from the center [1]. A +1 reward is awarded for every time-step the pole remains upward (non terminal step), and the pole is controlled by applying a discrete action, a force pushing the cart to the left or right (-1 or +1). The state space for the cartpole is given by

Nr	State	Min	Max
0	Cart position	-4.8	4.8
1	Cart velocity	-Inf	Inf
2	Pole angle	$-24°$	$24°$
3	Pole velocity at tip	-Inf	Inf

Table 3.1: State space Cartpole-v1

The Cartpole-v0 is defined solved when getting an average reward of 195.0 over 100 consecutive trials. In this project the environment is solved using the the REINFORCE-algorithm from [42]. This is a policy gradient method that solves the problem in about 600 episodes. The solution is stored in a checkpoint-file that is used when implementing the XAI-methods.

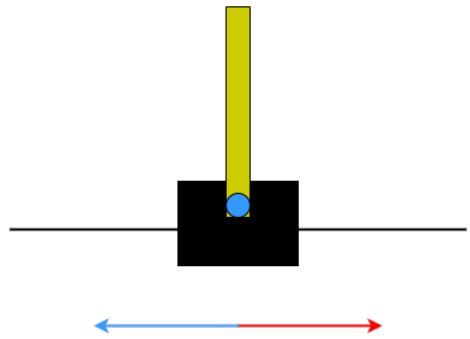

Figure 3.1: Cartpole schematic drawing

Lunar Lander v2

In the *Lunar Lander v2* environment the goal is to land a space-ship between two flags smoothly. The landing pad is always at coordinates (0,0), and the reward for moving to the landing pad at zero speed is about 100-140 point. If the space ship moves away from the landing pad it loses reward. The episode is done when the lander crashes (-100 points) or stands still (+100 points). It is also possible to get +10 points for each leg contact, and solving the challenge gives 200 points. Fuel is infinite, so an agent can learn to fly and then land on its first attempt, but firing main engine is -0.3 points each frame. LunarLander-v2 is considered "solved" when the agent obtains an average reward of at least 200 over 100 consecutive episodes [1]. The state and action space is given in table 3.2 and 3.3.

Nr	State	Nr	State
0	Horizontal position [x]	4	Angle
1	Vertical position [y]	5	Angular speed
2	Horizontal velocity [x]	6	First leg contact [bool]
3	Vertical velocity [y]	7	Second leg contact [bool]

Table 3.2: State space Lunar-Lander v2

The algorithm used in this thesis, deep Q-learning [44], solves the environment after

Nr	Action	Nr	Action
0	Do nothing	2	Fire main engine
1	Fire left engine	3	Fire right engine

Table 3.3: Action space Lunar-Lander v2

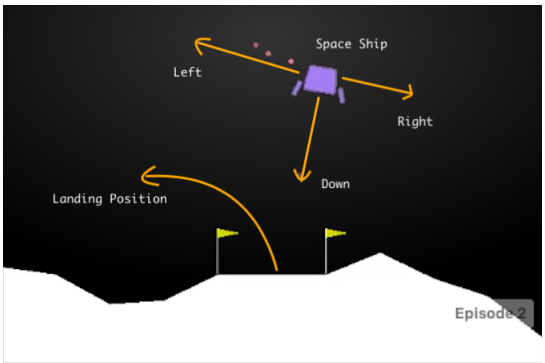

Figure 3.2: Lunar Lander schematic drawing from [43]

approximately 400 episodes. A checkpoint file is used here as well to implement the XAI-methods in a solved environment.

Robotic Manipulator

The third environment used in this project is a continuation from Sindre Remman's master thesis where a robotic manipulator (OpenMANIPULATOR-X by Robotis [2]) is trained using deep reinforcement learning [6]. The manipulator has four revolute joints and an end-effector, which means the total number of degrees of freedom is five. In one of the tasks, *reaching using DDPG*, the goal is to place the end-effector in the vicinity of a randomly selected point in the manipulator's workspace. If the end-effector is within a sphere with a radius of 2.0cm and origin in the randomly selected point, it is classified as being in the vicinity of the point. The points are randomly selected between episodes using randomized spherical coordinates. In [6], SHAP-values were used to interpret the decisions of a robotic manipulator. In this project this have been extended by comparing the explanations with the LIME method.

The manipulator consists of 12 states from positions and velocities of the 6 joints, the gripper is considered 2 joints because of how the simulated model works. The last 3 states are the goal coordinates, and they are concatenated with the joint states to create the whole state space of dimension 15. The action space is of dimension 4, where dimensions 1-4 corresponds to how much joints 1-4 should move in a single action [6].

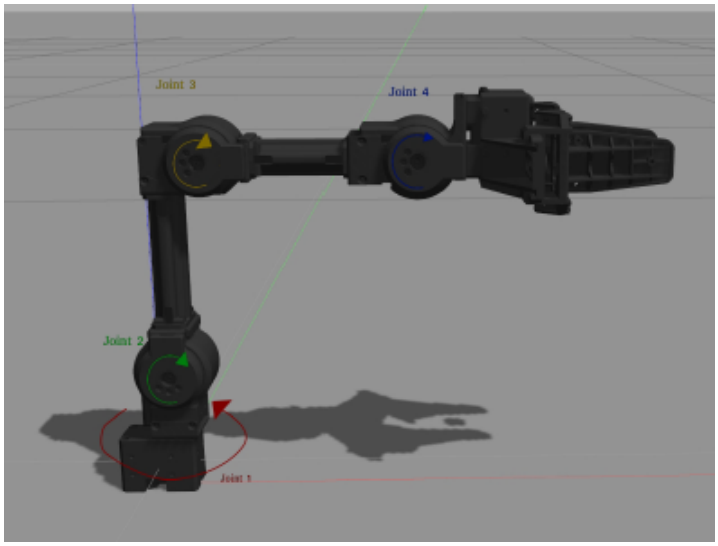

Figure 3.3: Robotic Manipulator from [6]

3.3 Methodology

The scope of this project is to implement XAI-methods across the three environments presented above, to investigate if these methods can produce plausible interpretations in robotic problems. The package of SHAP is implemented to produce both local and global interpretations, while LIME has been used to produce local interpretations in the same environments. In addition, to better understand the model's global behavior, Interpret ML has been used to compare the global SHAP interpretations with traditional feature importance techniques. SHAP and LIME are implemented using their respective packages from Github [3][4]. The documentation of the packages can be found here [45][46]. Interpret ML is also installed from a Github library [5].

The research structure is build upon an Anaconda environment with all necessary libraries activated, and the implementation is done in Jupyter Notebook with Python 3.8 and Pytorch. After training the environments with DRL-algorithms, checkpoint-files are saved, which has been beneficial so they can be used directly in further developments. A user-defined number of solved episodes are used in the XAI-methods. These are also saved with checkpoints, so they can be interpreted across multiple states locally and visualized by different plot functions.

The framework for this research was set up based on being as flexible as possible. One of the reasons for this was the uncertainty from the Covid-19 pandemic, meaning it was beneficial to be able working from home. Also, with the model-agnostic approach in the XAI-methods used, the explanation methods do not have a direct connection with the DRL-algorithms. This means the setup is extendable to other problems, and one change can be transferred between the environments.

An overview of how SHAP and LIME were implemented is presented below, including functions used from the libraries.

SHAP implementation

The SHAP DeepExplain function is used to get an explainer object. The function takes a neural net model from Pytorch together with a background dataset from the trained

episodes. The explainer object is used in the ShapValues function together with the test state to approximate the SHAP values for the deep learning model. This can be done locally with one state or over multiple episodes.

Three plot types from SHAP are used in this project.

- **Summary plot** - Shows a summary of the feature importance, how much impact each feature has on the model output across the different actions. The parameters are the SHAP values, a numpy array of the features and a parameter for how many included features.
- **Decision plot (Fig 3.4)** - Each plotted line explains a single model prediction. The function takes in a reference (base) value that the feature contributions start from, this is the expected value from the DeepExplain function. The base value depends on the size and distribution of the training dataset from the solved episodes. In the decision plot it is shown as a vertical line where each feature pushes the prediction away from the line according to the model output impact. The SHAP values and the test state are also inputs here. The logit operator is used to convert the output values from log-odds to probabilities.

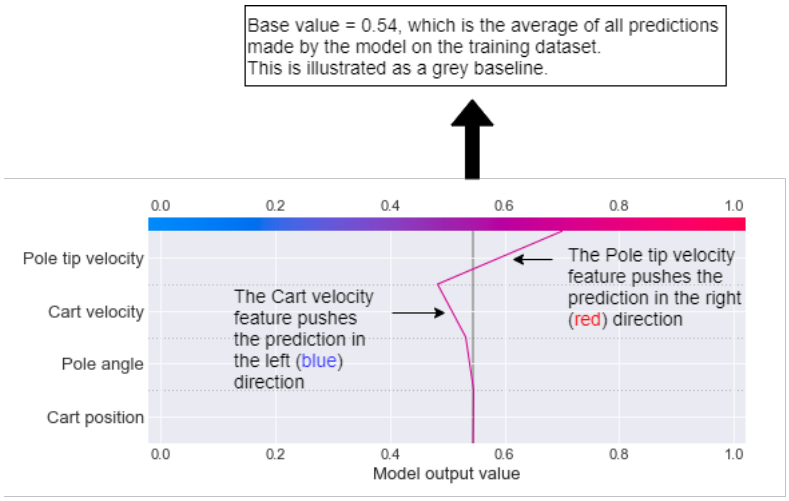

Figure 3.4: Example of a Decision plot. The red line is the prediction line of how each feature affects the prediction. Since this scheme gives an right prediction (>0.5), the line is red. Otherwise, it would be blue.

- **Force plot (Fig 3.5)** - The force plot is very similar to the decision plot, but visualizes the predictions in an additive way. Also here the expected value is used as a base value, and *"arrows"* are used to show if the predicted value is pushed higher or lower. The bold number on the number line shows the predicted value. The base value is the average model output over the training dataset used. Features contributing to pushing the prediction higher are shown in red. Features pushing it lower appear in blue.

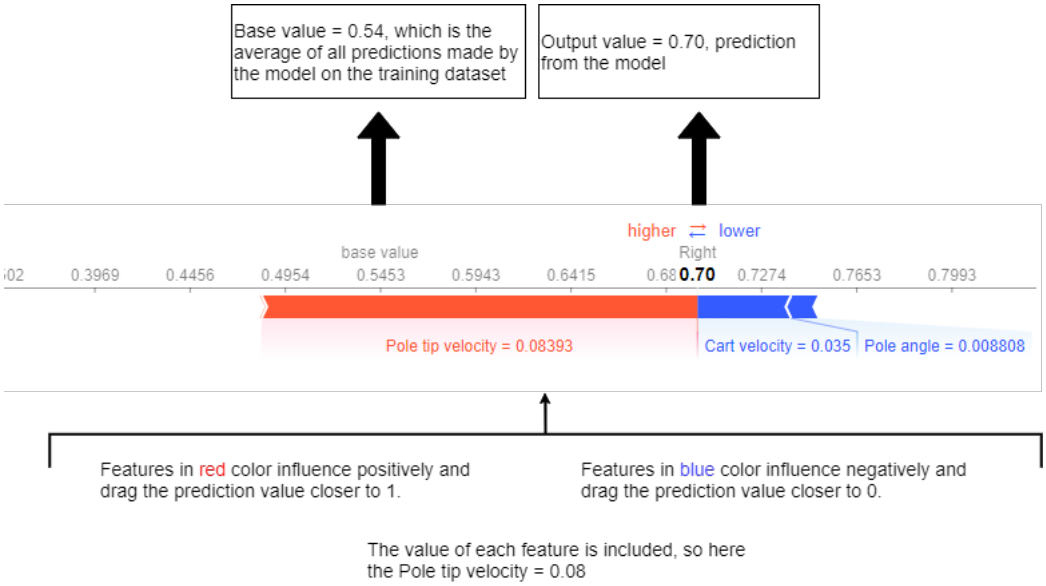

Figure 3.5: Example of a Force plot

Lime implementation

The LimeTabular function has been used from the LIME package, and the function takes a background training dataset to produce a LIME explainer object. This function also has some hyperparameter adjustments available. The kernel width adjusts the exponential kernel of the neighborhood sampling. In this project the author ended up with using the standard value, $\sqrt{col_{nr}} * 0.75$. It is also possible to test different discretizer options. In the end it was decided to use quartiles, as it gave the most stable and plausible results.

This means all non-categorical features will be discretized into quartiles [46].

The explain instance function is used to generate explanations for a local prediction by using the explainer object. This function generates neighborhood data by randomly perturbing features from the test state which is one of the parameters. This is used to learn locally weighted linear models on this neighborhood data to explain each of the classes in an interpretable way. The function uses a prediction function which outputs prediction probabilities for a given state. This is defined according to the algorithms used, and is different between the environments. A soft-max operator is used to normalize the predictions and in Lunar Lander a convolution was also implemented in this function. Defining the prediction function across three different environments was one of the challenges encountered in this project. An explanation object is returned with the corresponding explanations, and can be visualized showing prediction probabilities and feature importance for the local prediction.

Interpret ML

Interpret ML was implemented in the Cartpole environment when searching for methods to compare the global SHAP results. InterpretML is an open-source package that incorporates state-of-the-art machine learning interpretability techniques under one roof [5]. The Morris sensitivity and Partial dependence plots used from this package also take a prediction function and a training dataset to produce global interpretations of features.

Chapter 4

Results

In this chapter, results will be presented from all three environments. The plots are from the packages of SHAP and LIME, which means that some of the visualizations can be a bit unique compared to traditional methods. Section 3.3 gives an overview of the methods used and explanations about how the plots can be interpreted.

4.1 Cartpole-v1

Local explanations

Cartpole is the simplest environment used in this thesis, with four states and only two actions. LIME and SHAP were implemented on solved episodes from a DRL model, and used to provide explanations. In the first example, only one solved episode was used to train the explainers, meaning approximately 300 states. One local situation is used to visualize local explanations, shown in Fig 4.1. The key for Cartpole is using inertia and acceleration to balance the pole. In this example the Cartpole has a positive (right) pole tip velocity, meaning the intuitive action can be to push the cart to the right to stabilize the pole in the other direction. However, the cart velocity is also in positive direction, meaning a left push action could also be considered to slow the cart down. LIME explanations are shown in Fig 4.2. The LIME output consists of prediction probabilities, explanations comparing actions and the feature values in the example state.

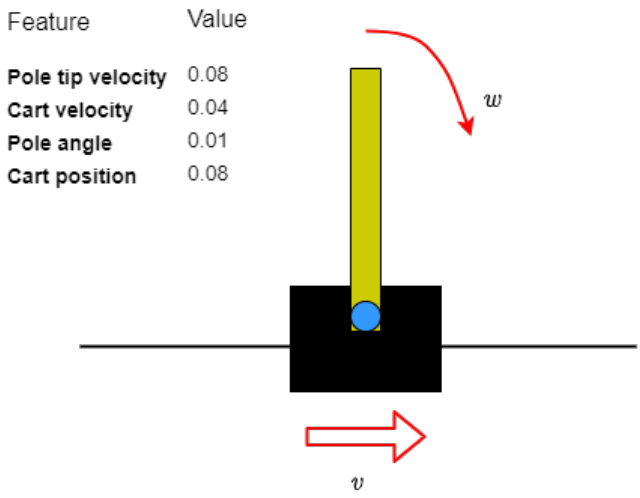

Figure 4.1: Cartpole: Schematic figure for Example 1 with feature values

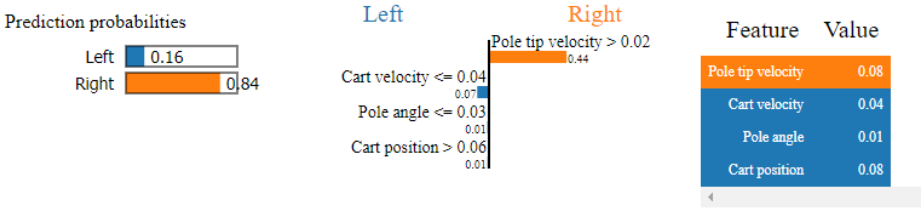

Figure 4.2: Cartpole: LIME explanations for Example 1

The local SHAP explanation is illustrated by a force plot (Fig 4.4) and a decision plot (Fig 4.3). The decision plot shows how each feature pushes the explanation to the right or left, while the force plot shows the impact and prediction probability of the most important features. In the Force plot, the red "arrows" illustrates a push to the right and the blue "arrows" illustrates a push to the left. DeepSHAP uses the model instead

of a prediction function, which means the explanations are presented in SHAP value magnitude, but here the logic operator is used to transfer these values from log-odds into probabilities. The base value in both SHAP plots are 0.55, which is the expected value of choosing a right action over all episodes. A base value at a distance away from 0.5 can indicate that too few states are used to train the explainer.

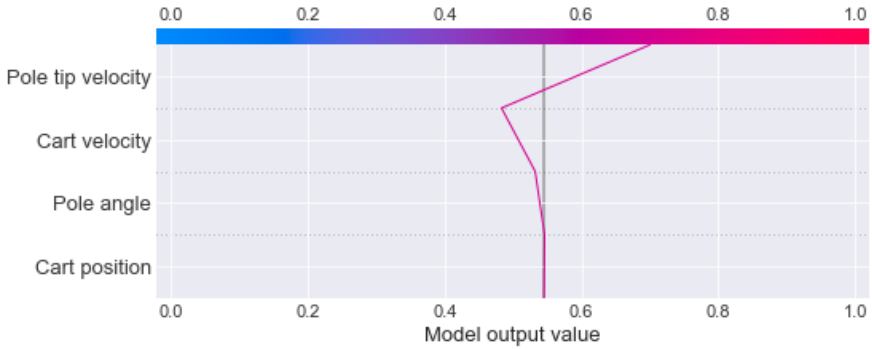

Figure 4.3: Cartpole SHAP Decision plot example 1. Baseline 0.55, red color means action right while blue color action left

Figure 4.4: Cartpole: SHAP Force plot example 1. Red "arrow": push to the right, blue: push to the left

As seen in Fig 4.4, SHAP proposes the action of pushing the cart to the right, but not as much as LIME in Fig 4.2. Both methods highlight the same features in the same directions, which also makes sense when looking at the feature values of the example state. However, the difference in the prediction probability is harder to explain, but it could be because of the approximations in DeepSHAP or the neighborhood sampling of the LIME method. This will be discussed further in the discussion in Chapter 5.

The second example is more even, meaning the feature values are lower and it is not easy to choose a clear action. In this example, 5 episodes were used to train the explainers, and the base value is closer to 0.5 when using 1500 states. While a left push could make sense with a small negative pole angle and pole tip velocity, the cart is already going in the left direction and a right push must also be considered to slow it down. The example state is illustrated in Fig 4.5.

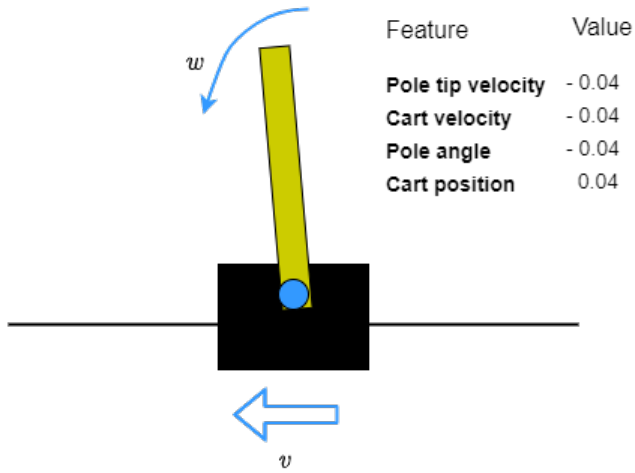

Figure 4.5: Cartpole: Schematic figure for Example 2 with feature values.

SHAP and LIME highlight the same features in similar directions here too. While LIME predicts a probability of 0.53 of pushing the cart to the right, SHAP is a bit lower with 0.51. This was the common pattern when testing these two methods in the Cartpole environment. Both methods were giving mostly similar predictions, but SHAP tending to be a bit more conservative in the estimations.

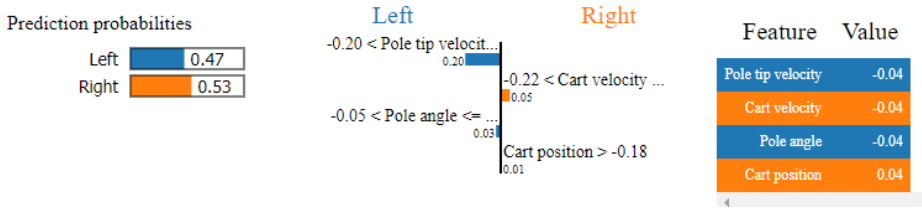

Figure 4.6: Cartpole: LIME explanations for Example 2. Note: The values in the left/right plot do not necessarily imply the prediction probabilities as other factors are also considered.

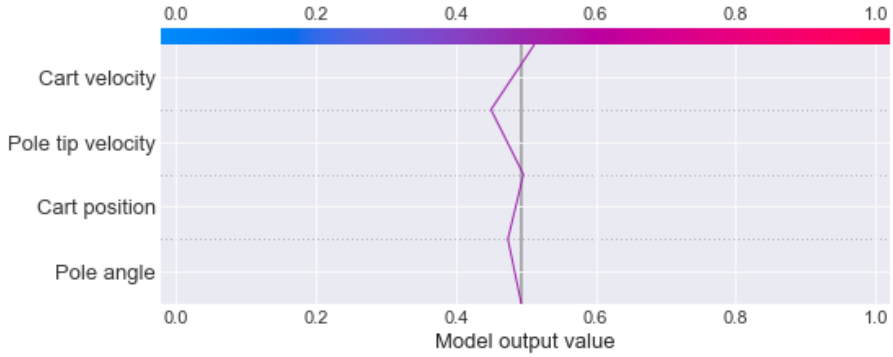

Figure 4.7: Cartpole: SHAP Decision plot for Example 2

Figure 4.8: Cartpole: SHAP Force plot for Example 2

Global explanations

To produce global SHAP explanations, data was collected from 5 solved episodes. This means approximately 1500 states are used when training the SHAP explainer, and calculating global SHAP values took 2-3 minutes on the school computer provided with 32gb of RAM. The global explanation highlights the most important features over these solved episodes, and the summary plot (Fig 4.9) shows how much the different states contribute to the magnitude of the different actions.

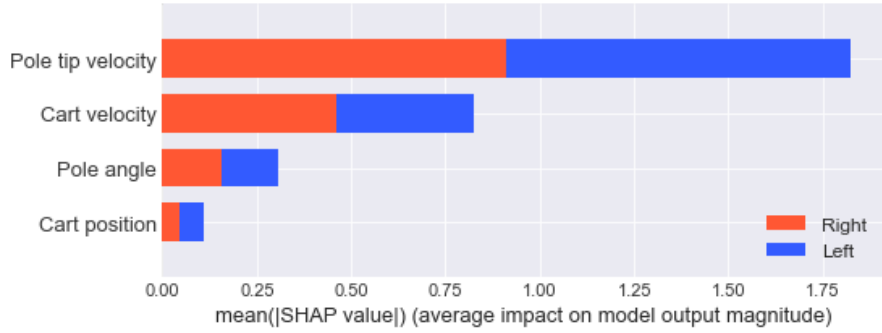

Figure 4.9: Cartpole: SHAP Global Summary plot over 5 episodes

The global summary plot shows that the two velocity features have the biggest impact on the model with the actions almost evenly distributed in each feature. When stabilizing a Cartpole, it makes sense that possible changes in the velocity features must be corrected immediately to keep the pole in an upright position. However, such a summary plot also produces more questions. Even though the magnitudes make sense from a practical view, it is difficult to be certain about these results when the basis of comparison is limited. Fortunately, Interpret ML is providing a package that includes traditional sensitivity methods that can help gain some insights about these global explanations. This was implemented in the Cartpole environment, and the global summary plot can be compared to the Morris sensitivity index (Fig 4.10).

The Morris plot in Fig 4.10 measures the sensitivity mu_{ij}, described in Section 2.7. The method summarizes variation effects to estimate global sensitivity in the output

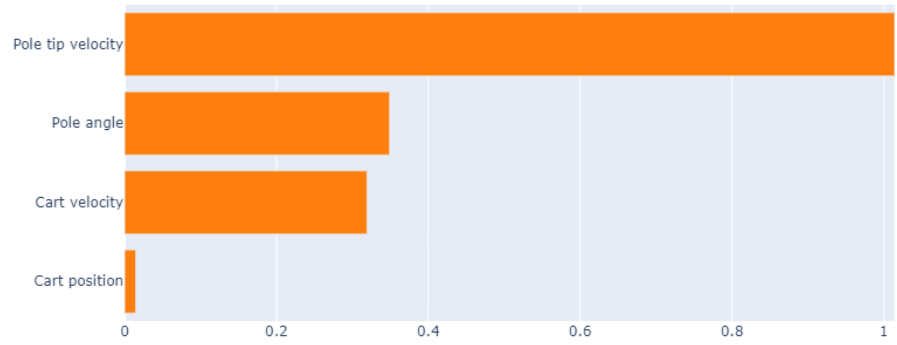

Figure 4.10: Cartpole: Global Morris Sensitivity plot over 5 solved episodes with the sensitivity index mu_{ij} on the x-axis

space, and can be helpful when testing the robustness of the model or understanding the relationships between the input and output variables. In this case it can be used to get an understanding of the global SHAP magnitude predictions, will uncertainty in one feature be transferred into the perturbations of the SHAP method?

Both methods show that the pole tip velocity by far is the most important feature in the Cartpole environment, while the cart position has least impact. However, the pole angle has a much higher relative value in the Morris analysis compared to SHAP. This can indicate that SHAP are not taking the pole angle into enough consideration. But it is also a possibility that this shows some of the drawbacks of comparing these two methods. An uncertainty analysis of the pole angle can lead to high changes in output values. When it differs from zero, the model must stabilize. This is not necessarily the case for these solved episodes, where the pole angle is mostly very small. This can be illustrated with a partial dependence plot of the specific features (pole angle and cart velocity).

These PDP-plots are another way of showing the marginal effect one feature has on the predicted outcome of a machine learning model. Fig 4.11 shows the average response and the value density over the 1500 states for the pole angle, while Fig 4.12

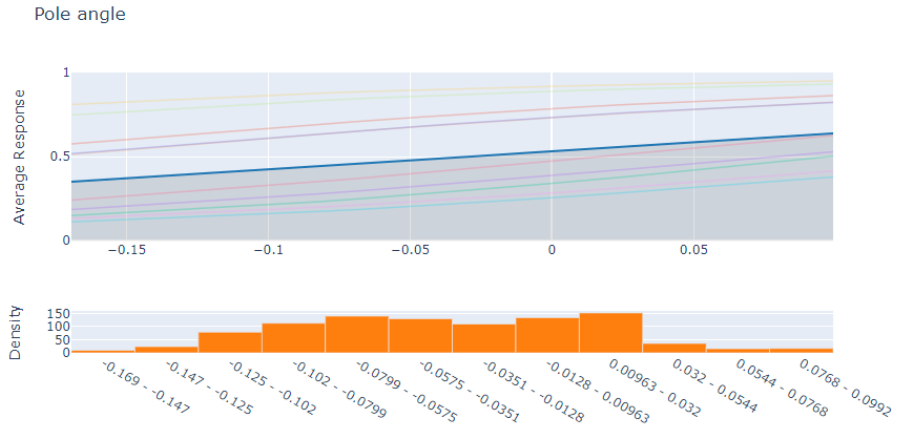

Figure 4.11: Cartpole: Partial dependence plot **Pole Angle**. Top plot: changes in feature value on x-axis. Bottom plot: actual feature values compared to the density over 5 episodes.

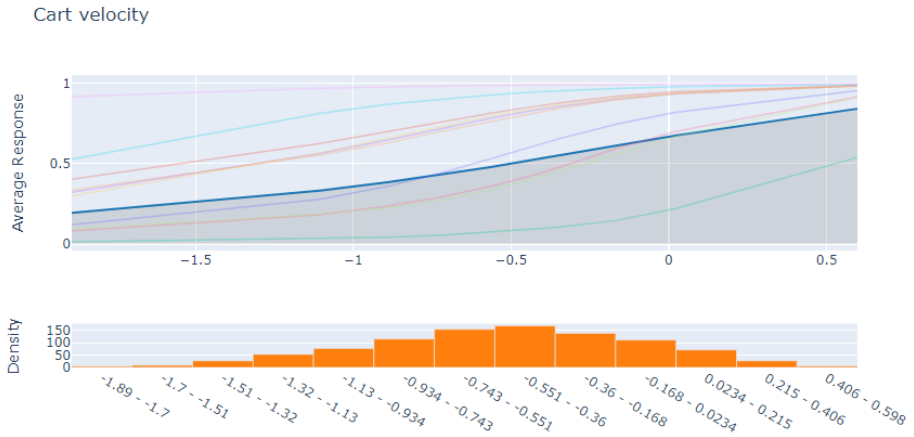

Figure 4.12: Cartpole: Partial dependence plot **Cart Velocity**. Top plot: changes in feature value on x-axis. Bottom plot: actual feature values compared to the density over 5 episodes.

does the same for cart velocity. The blue line in the partial dependence plot is the average response, while the individual response lines illustrates the standard deviation outline. From the density estimation, it is seen that the feature values of the cart velocity are much bigger than the pole angle, and the average line is also steeper. This could possibly support the SHAP analysis, and shows that over these 1500 states the pole angle is relatively small and does not have the same impact as in the uncertainty analysis.

To illustrate the model output value of the solved episodes, a global decision plot (Fig 4.13) was created. As with the local explanations, when the model output is small (< 0.5), the model predicts a left push action, while a model output (> 0.5) predicts a push to the right. This plot illustrates the SHAP explanations both locally and globally. A red feature feature density refers to a positive (right) value, while a blue one refers to a negative (left) value. A high pole tip velocity in the right direction means the model must push to the right to stabilize the pool, while a negative cart velocity also means a right push to slow the cart down. The global decision plot connects local and global explanations, and confirms that the features with greatest SHAP-magnitude in the summary plot have the biggest impact on the model output.

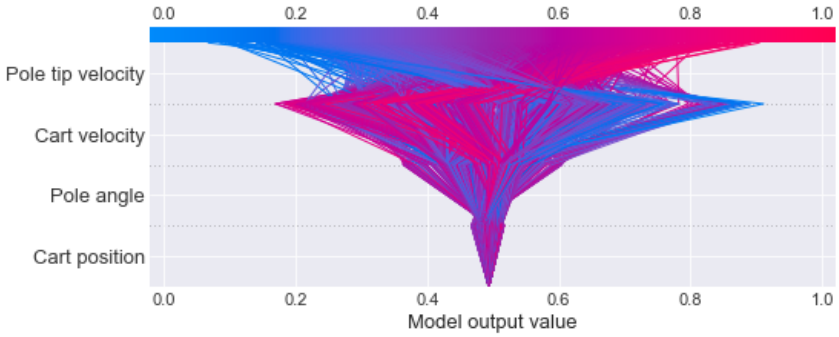

Figure 4.13: Cartpole: SHAP Global Decision plot

4.2 Lunar Lander

The Lunar Lander environment consists of 8 states and 4 actions. This means it is more challenging to visualize the explanations, and many different factors influence the decisions taken by the model. In the following section local explanations for one example state will be presented together with global explanations from SHAP. Both explanations are trained on 5 solved episodes, which means approximately 3000 states are used during training.

Local explanations

The example state shows the Lunar Lander in the middle of a landing operation (Fig 4.14), where the lander is a bit to the left of the yellow flags. To manage the landing between the two yellow flags, it must move slightly to the right. Even though a left action can be the intuitive choice here, other actions must also be considered to stabilize the lander. It is already tipping a bit to the right and a left action could trigger this further, and in the worst case cause a crash. It could also be possible to fire the main engine or do nothing in this certain state and wait until the lander gets closer to the ground.

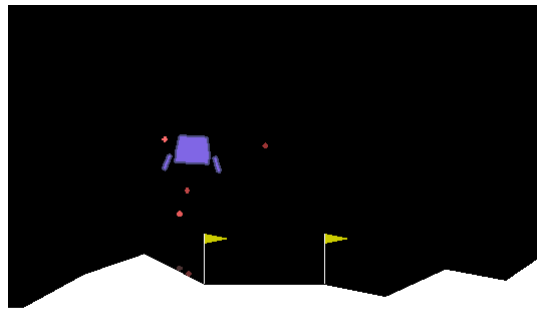

Figure 4.14: Lunar Lander: Example scheme

The prediction probabilities shows an almost even distribution between the four actions. A fire left action, which steers the lander to the right, is slightly preferred, but the differences are marginal.

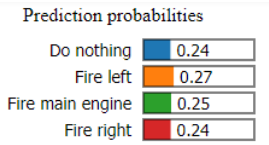

Figure 4.15: Lunar Lander: LIME prediction probabilities

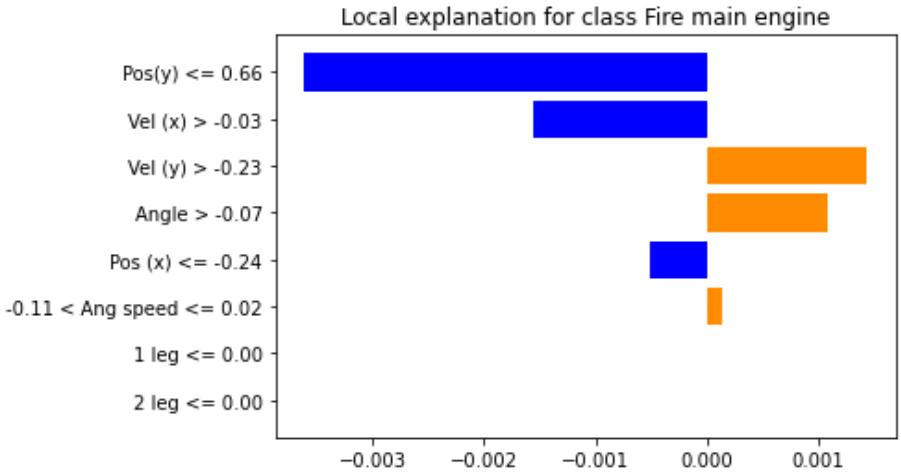

Figure 4.16: Lunar: LIME plot for action **Fire Main Engine**

With four actions it is more difficult to separate the explanations. Both SHAP and LIME are giving very small values in this environment, which makes it challenging to trust the explanations since small changes can have a big impact on the output. However, both LIME (Fig 4.16) and SHAP (Fig 4.17) mainly agrees on which features that are most important. These figures show how each feature pushes the prediction of the action fire main engine in the positive (red/orange) or negative (blue) direction. This was also the case for the other 3 actions, but disputes between the two methods were present for all actions. The methods agree that Pos(y) and Vel(y) are the most important features in each direction. However, LIME highlights angle as an important feature while it is not present in the SHAP explanation. The same yields for Vel(x) in the opposite direction.

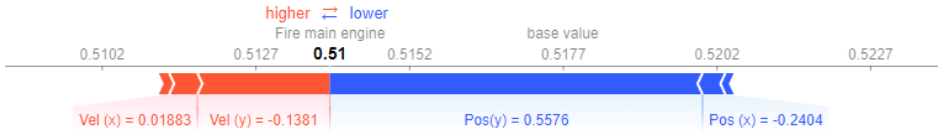

Figure 4.17: Lunar Lander: SHAP Force plot for action **Fire Main Engine**

To illustrate the differences between the features in this local situation, a SHAP summary plot is included. This shows the local SHAP magnitude for each feature and action in this particular state. A big contribution from the velocity and position features are expected, and since the legs do not have any contact to the ground in this example these features should be zero. However, the low SHAP values can indicate vulnerability in these explanations, which leads to a uncertain reflection about the true behavior of the model.

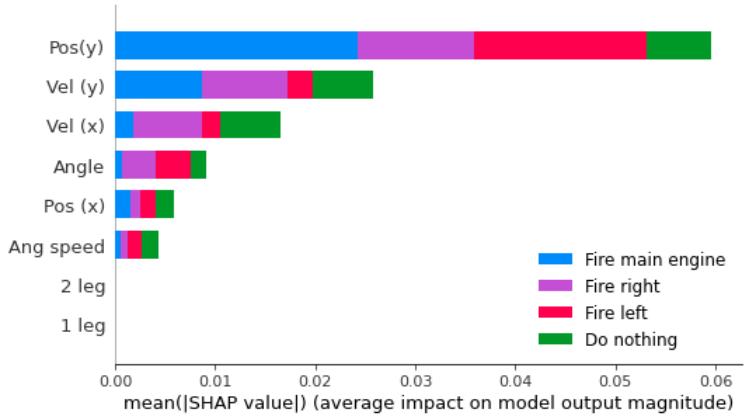

Figure 4.18: Lunar Lander: SHAP Local Summary plot

Global explanations

Calculating the global SHAP values took 10-15 minutes, and the resulting summary plot is shown in Fig 4.19.

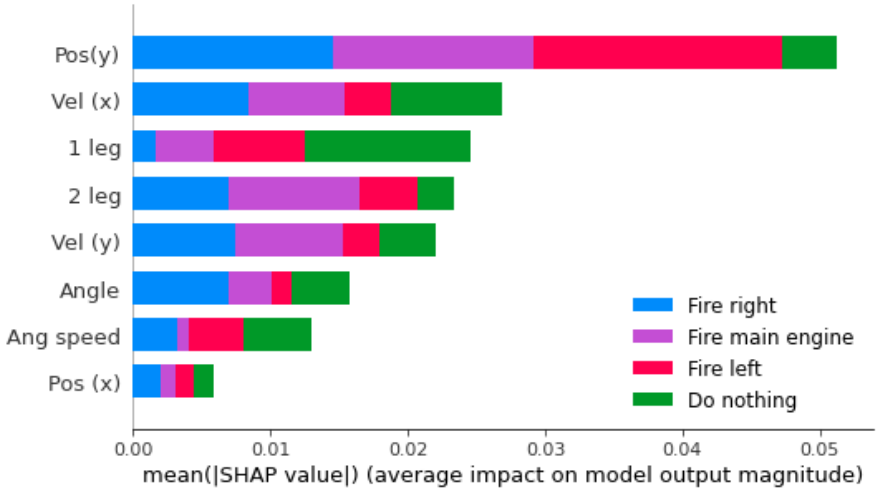

Figure 4.19: Lunar Lander: SHAP Global Summary plot

The global plot has many similarities compared to the local explanations. The most noticeable difference is the leg-features, as they have a big impact in a global sense when the Lunar Lander is touching the ground. However, this plot also shows some weird results from the model. It is strange that the position in X-direction has so little impact. This is usually a pretty dominant feature when landing between two flags. The big differences between fire left and fire right in some of the features could also indicate that too few episodes are used when training these global explanations, but the amount is well within what is recommended. As with the local explantions, low SHAP values can be a sign that the explainer does not reflect the true behaviour of the model. The challenges of explaining the Lunar Lander environment will be discussed further in Chapter 5.

4.3 Robotic Manipulator

Local explanations

In Sindre B. Remman's master's thesis SHAP values were used to interpret results on a robotic manipulator with 15 states and 4 actions. In this project these explanations have been extended with more plots and also by adding LIME explanations. In the situation used the agent is just a single step away from the goal. This is shown in Fig 4.20, and the agent only needs to choose the action $[1.0, 0.0, 0.0, 0.0]$ to reach the goal, which corresponds to moving the first joint 0.2 rad. The SHAP explainer was trained on 10 000 randomly selected poses and goal in vicinity of this situation to produce the local explanation shown in Fig 4.21.

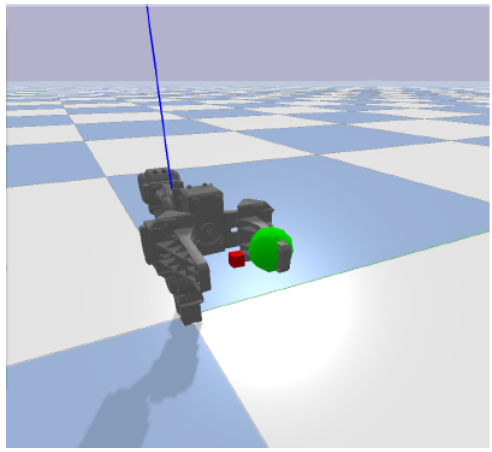

Figure 4.20: Robotic Manipulator: Situation 1. The red box in the middle of the end-effector is one step away from the goal (green)

The local SHAP explanations show that the position of joint 1 is the most important, and by far the most contributing to how joint 1 should be changed. This corresponds to the constructed situation of being one first joint step away from the goal. An interesting observation is that the position of joint 1 also is important for how joint 3 should change. This could maybe imply some challenges with the agent, since LIME prediction probabilities also gives a much higher probability of choosing joint 3 than expected (Fig

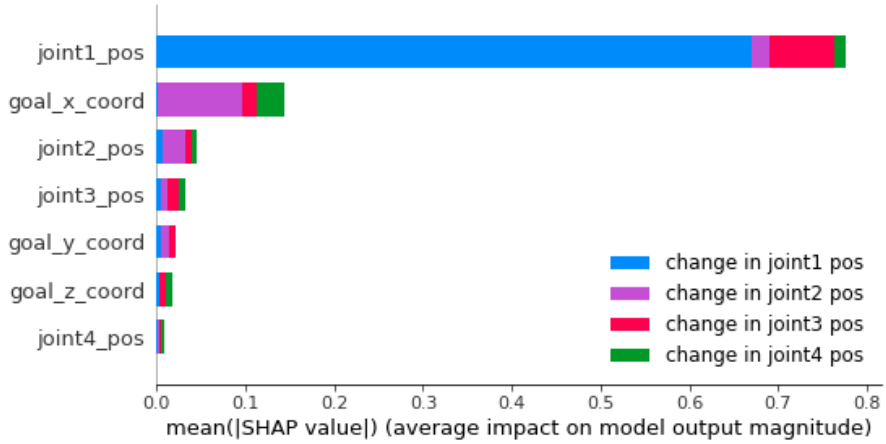

Figure 4.21: Robotic Manipulator: Shap Local Summary plot

4.22) [6].

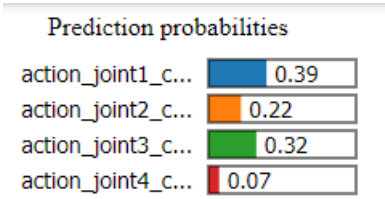

Figure 4.22: Robotic Manipulator: LIME prediction probabilities

Local explanations for joint 1 and 3 are included to show this phenomena. Both the LIME explainer (Fig 4.23) and the SHAP force plot (Fig 4.24), shows that the joint 1 position is the crucial factor for changing the first joint.

However, the explanations for change in joint 3 position shows something interesting. A lot of the features are pushing for a joint 3 action in both explainers. With LIME (Fig 4.25) some other features push against this action while the SHAP force plot (Fig 4.26) only shows features pushing in the direction of changing joint 3. In addition, the base

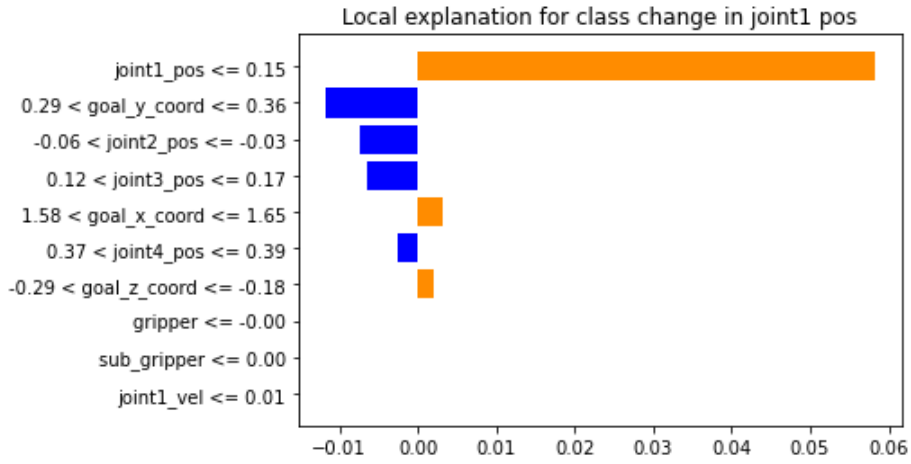

Figure 4.23: Robotic Manipulator: LIME explanations **Joint 1**

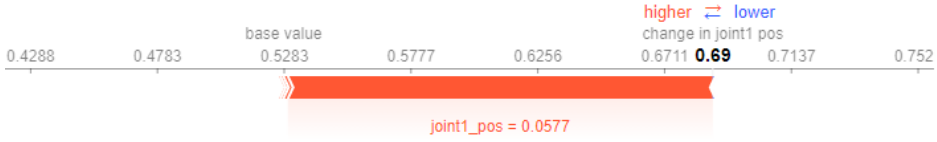

Figure 4.24: Robotic Manipulator: SHAP Force plot **Joint 1**

value for this action is high, meaning a joint 3 action could be over-estimated in the training set of randomly selected poses.

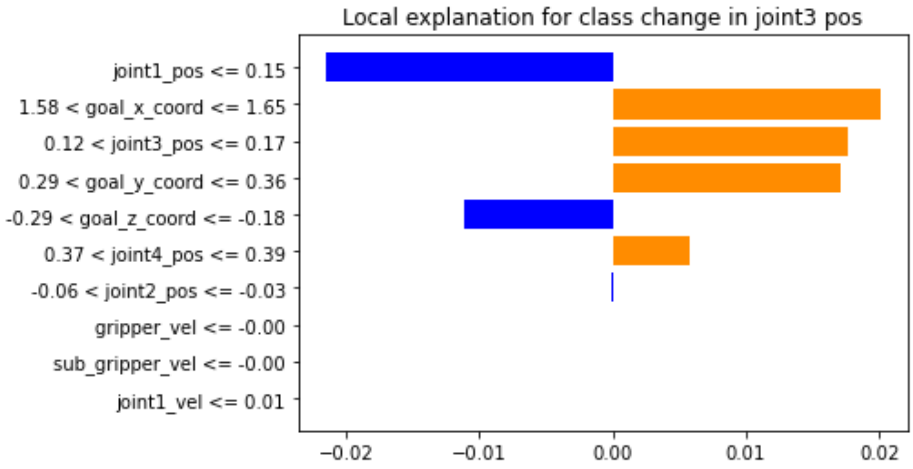

Figure 4.25: Robotic Manipulator: LIME explanations **Joint 3**

Figure 4.26: Robotic Manipulator: SHAP Force plot **Joint 3**

Global explanations

The global SHAP explanations in Fig 4.27 are showing a summary of how much the different sates contribute to the magnitude of the different actions. These are collected by training the SHAP explainer on 5000 randomly selected poses and goals, but here the SHAP values are calculated across the dataset instead of a chosen test state. Because of the computing power needed to calculate the global SHAP values, the dataset size was decreased from 10 000 used in [6], but it still took over three hours.

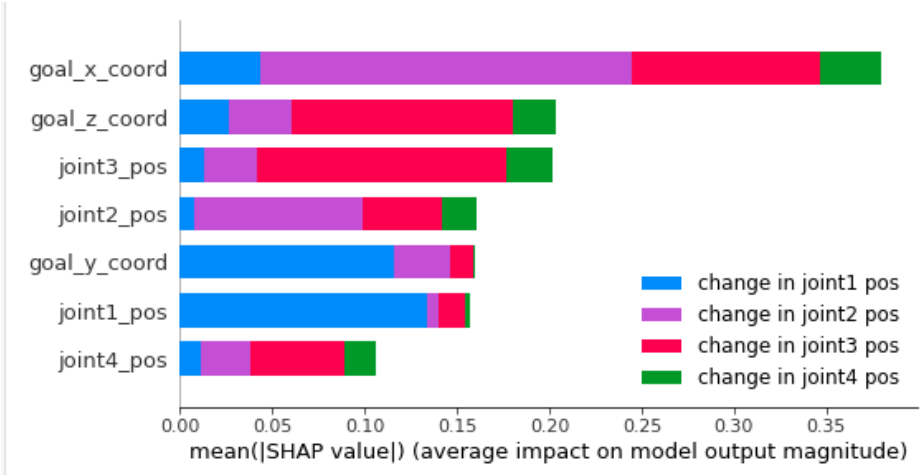

Figure 4.27: Robotic Manipulator: SHAP Global Summary plot

As stated by Sindre B. Remman [6], the global result shows something reasonably disconcerting. The z-position of the goal is relatively important for how joint 1 should be changed, but this should not be the case since a joint 1 rotation around the base will not influence the z-coordinate of the end-effector. These global SHAP explanations can be used to evaluate the agent. Such discoveries decrease the trust of the agent, but it could also be because of assumptions being made in the SHAP explainer.

Some of the patterns and interpretations from the explanations across these environments will be discussed further in the next chapter.

Chapter 5

Discussion and analysis

The task of improving robots' autonomy has been a goal for many decades. Recent advances in AI make room for controlling robots with DRL, which make them able to learn from experience and improve performance over time. XAI is also an emerging field that aims to let AI technology be better understood by humans. Combining these two has been challenging, with small amounts of existing research. The problem with DRL is that no clear answers about the model's decision exist. In many ways, it is comparable to human reasoning, and interpretation is based on intuition. In supervised learning, where most of the XAI research is taken place, it is easier to say something about the explainers' performance when it can classify a correct or false prediction.

In this project three DRL-models have been used to explore the possibilities of which interpretations some of the most promising XAI methods can do in robotic environments. The results in Chapter 4 shows that the methods produce quite similar explanations across all three environments. From the author's perspective, the results in the Cartpole environment often coincides with the predictions an human would have made in the same situation. When increasing the dimensionality to more complex environments, the results are more unstable and difficult to interpret. This is especially present in the Lunar Lander environment where the prediction probabilities are evenly distributed and the SHAP magnitudes are small. Not much information can be extracted with those results, and the amount of trust gained about the model is minimal. On the robotic

manipulator, the explanations are again more in line with the author's predictions. The explainers give signs of possible issues that decrease the trust of the model. However, multiple issues with the XAI methods used in this project could also amplify disconcerting factors of the agents' when the problems just as well could arise in the explainers.

From the results in this project and literature available [38][47][48], a comparison between the two methods have been made.

5.1 SHAP

- The SHAP framework is based on solid theory. The properties from game theory give the algorithm a reasonable foundation, and the efficiency axiom makes the difference between the prediction and average prediction fairly distributed among the feature values. This could emphasize the fact that SHAP results were more stable throughout these experiments. It could also explain the differences between local Example 1 and 2 in Cartpole, which showed that training the local examples over multiple episodes gave more reasonable base and prediction values. By using all the features when doing the explanations and distributing the effects fairly, the framework could be a more legally compliant method, but it arguably lacks some of the user-friendliness LIME has.

- The Shapley estimation can be misinterpreted. The SHAP magnitude explains the contribution of a feature value to the difference between the actual prediction and mean prediction. It returns a magnitude value per feature, but no prediction model like LIME. In this project, these values have been converted into prediction probabilities and visualized in a way that is comparable to the LIME predictions using the logit operator. However, because of the approximation in DeepSHAP, this is not necessarily the correct prediction probabilities. The weighting when getting the SHAP values for local explanations could be affected by this, and even though it fulfils the Shapley value properties, it could miss some of the true behaviour of the model in the local neighbourhood. This could explain some of the differences between SHAP and LIME in the previous chapter.

- SHAP is able to produce both local and global explanations. This is a huge benefit

when analyzing deep learning models, both local and global insights could be very useful. For neural network models, DeepSHAP is recommended over other SHAP methods, for example KernelSHAP, because of much faster computation time. From the experiments in this project, local SHAP values are computed very fast, while global SHAP values demanded a lot of time. For local approximations, KernelSHAP would probably produce more similar predictions compared to LIME. KernelSHAP is based on a combination of LIME and Shapley values and does not approximate SHAP values by a linearization. On the other hand, this project have been useful for understanding what this approximation means when comparing the local predictions.

- The assumption of feature independence is the elephant in the room of this project. Shapley value method suffers from inclusion of unrealistic data instances when features are correlated. Marginalizing the features is fine as long as the features are independent. When they are dependent, feature values that do not make sense, can might be sampled. Fig 5.1 shows the correlation over 5 episodes in the Cartpole environment, while Fig 5.2 does the same for Lunar Lander. These figures show that the correlation is generally much higher in the Lunar Lander environment, with multiple feature pairs well above ± 0.75. Cartpole also has high correlation between some of the features, but none above ± 0.6, and is generally at a lower correlation level.
  A common problem with SHAP is that collinear features could assign a SHAP importance score close to zero, which can be the reason for the low magnitudes in the Lunar Lander environment. This assumption is problematic in the field of XAI because it can cause wrong explanations. In robotic problems, this is a big drawback, since the features are naturally correlated. Some solutions to this problem have been proposed, such as permuting correlated features together, but this could violate the symmetry axiom in the Shapley estimation. The assumption of independence also yields for partial dependence plots.

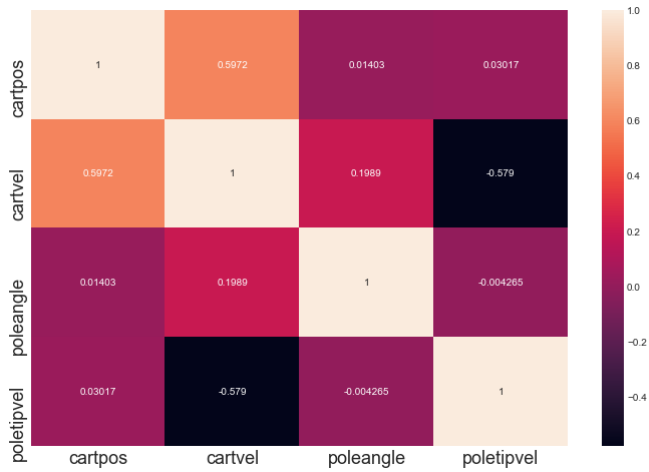

Figure 5.1: Cartpole: Correlation matrix over 5 episodes. Values closer to ±1 are more connected (high correlation)

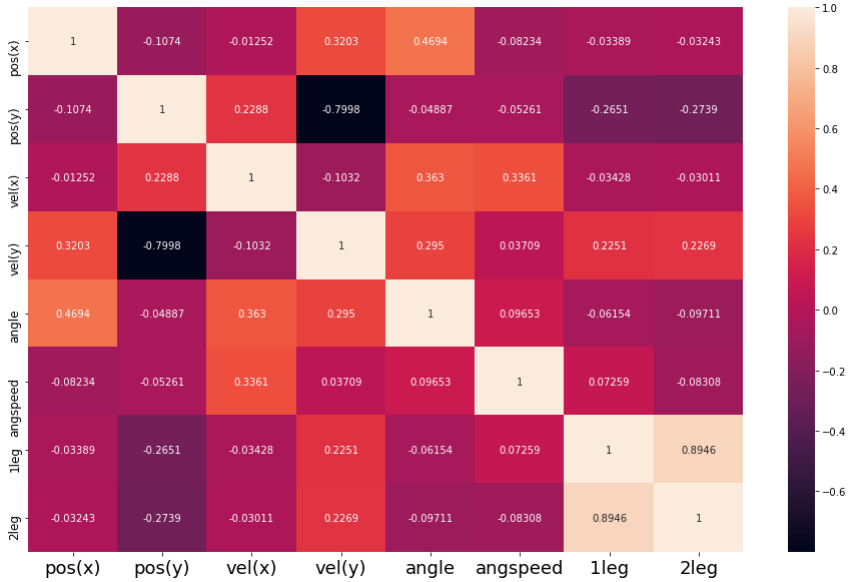

Figure 5.2: Lunar Lander: Correlation matrix over 5 episodes. Values closer to ±1 are more connected (high correlation)

5.2 LIME

- LIME is easy to implement and use across different models, and works for multiple data types. The interpretations given are easy to understand, it gives clear prediction probabilities together with feature importance. It does not say anything about magnitudes or complete attributions, but gives easy human-friendly explanations. LIME can only do local predictions, but it calculates these very fast. In this project all of the explanations, even the more complex ones, were predicted within a few seconds.

- The correct definition of the neighborhood is a very big, unsolved problem when using LIME. This is especially important with the environments that use tabular data. Samples near the point of interest are being weighted more heavily than samples far away, and when using discretization it is not possible to differentiate within the discretized bins. The way sampling is done in LIME can lead to unrealistic data points, and the local interpretation can be biased towards those data points. Some of this can be accounted for by tuning parameters to adjust the neighborhood, and this was tested with the kernel width and discretizer options in this project. However, this could affect the interpretations of the model and be use case specific.

- As with SHAP, the correlation between features is ignored since sampling is from a Gaussian distribution. Another problem being reported with LIME is the instability of the explanations. This was present in this project, explanations of similar examples could suddenly be totally different. The lack of robustness and assumption of feature independence decreases the trust of the method, and put the explanations under a critical view.

5.3 XAI or XRL?

The reason model-agnostic explainable methods (LIME and SHAP) have been used in this project is because they are decoupled from the underlying machine learning model. This makes them easier to implement across multiple environments and gives explanation and representation flexibility. Because of these scaling abilities, they are

expected to be a dominant factor in the XAI-field [49]. However, when using complex models based on deep neural networks, such as the ones in this thesis, other methods could be considered. Tools to uncover features and concepts directly in the hidden layers could possibly be more effective. An example is Integrated Gradients who utilizes the gradient in the model instead of looking at the model *from the outside* [50].

In hindsight, the issues about these model-agnostic methods could produce doubt about if these methods should be used to explain Deep Reinforcement Learning in the first place. The assumption about feature independence decreases the trust in the explanations, and especially the Lunar Lander environment shows how highly correlated features affect the methods. It may be to early to expect that combining DRL and XAI will increase the trust in the models, when these frameworks are still mostly in the research phase. It would also be very helpful to have a universal design language to visualize the explanations. This would make the development phase easier, and could convey the results in a much better way. During the work on this project, research about explainability in DRL has started to get more attention. Assessing how XAI techniques can understand models beyond classification tasks has not been extensively studied before, but a new subfield in Explainable Reinforcement learning (XRL) is emerging [51].

The most important thing in XRL is to keep the human side of the equation in mind [49]. No clear answers make interpretations more difficult, and explanability should possibly be used as a tool instead of a solution. The results in this project shows that promising XAI techniques could help interpret the black-boxes in DRL and give explanations that makes sense from an human perspective. In simple environments like Cartpole where the features are not too correlated, this can be used as support to understand the model, both in a local and global scope. This shows that XAI could enlighten insight about the deployment of everyday problems in the RL-community. However, when the environments get more complex with highly correlated features, some of the challenges by using XAI methods in DRL are much more present.

Chapter 6

Conclusion

6.1 Answering the research questions

Can methods from Explainable Artificial Intelligence (XAI) be used to interpret the results on models trained with Deep Reinforcement Learning (DRL)?

The goal with XAI methods is to open up the black-box problem of neural networks to interpret decisions made by the agents. One of the drawbacks of robotic reinforcement learning is that mistakes in such systems could lead to dramatic consequences, and therefore the amount of transparency must increase before it gets more adapted in the industry. The methods used in this thesis show that XAI could help increase the transparency in simulated robotic problems trained with DRL. It can give information about the agent's predictions and important features, both in a local and global scope. The problem is that these predictions do not have any clear basis of comparison, in contrast to machine learning problems where it is possible to classify a correct prediction. This makes the XAI methods more challenging to visualize and convey, it depends a lot more on human intuition.

The limitations and assumptions within the XAI methods used in this thesis also show some of the challenges by using these methods in complex problems with highly correlated features. It is difficult to interpret these systems when we do not know if

either the XAI methods' predictions or the agent could be trusted. From the author's perspective, these methods can be recommended to open up a bit of the black-box. But they should be used in a critical sense together with other tools to get more transparency.

Of the two most common XAI methods today, LIME and SHAP, which framework is best suited to robotic applications in real-life environments? What are the important factors to consider when choosing between the two methods?

The challenges discussed above regarding correlated features make these methods vulnerable in robotic environments. When choosing between them, the choice is simple when searching for a global perspective, since SHAP is the only method to deliver global explanations. Locally, the choice is more dependent of the interpreter. The LIME method is simpler and arguably gives more human-friendly predictions. SHAP has a better theoretic foundation and a bigger framework for visualizing different results. Simplified, LIME tells you what is the most important attribute around the interest point. SHAP tells how you got your score, so the choice depends on the given use case. The decision could for example be taken depending on a couple questions the researcher needs to considerate.

- **Local or global scope:** Do you care about reasons for a specific decision or the whole logic of a model? Global SHAP, local see questions below:
- **Time limitation:** Does the user need to quickly take a decision? LIME could be preferred, since it is the fastest method and can produce human-friendly explanations within a few seconds. If there is no time constraint, SHAP could be preferred to get a more complex explanation. However, local SHAP explanations are also computed fast.
- **Expertise level:** What expertise level does the interpreter have? SHAP demands a greater understanding of the method and Shapley-values, while LIME does not demand much background knowledge.

6.2 Further work

This work will be continued in the spring of 2021, during the work on the author's master's thesis. The work done during this pre-project has laid the groundwork for the master's thesis both when it comes to the background theory used and familiarity with combining XAI and DRL.

An interesting perspective would be to explore methods that uncover features and concepts directly from the neural networks, and compare them to the model-agnostic approach in this thesis. Promising techniques are Integrated Gradients and Linear Model U-trees [50][52]. More methods from the XRL community can also be tested. This is a field where it is hard to find papers, so more research is needed. Luckily, progression is being made every month, and while writing this thesis a survey with options became available [51].

The next step will be to test these methods on real-life environments and determine how they can be used to improve these applications effectively. This project is a part of the EXAIGON project at NTNU which aims to meet the society's and industry's standards for deployment of trustworthy AI systems in social environments and business-critical applications by developing methods for understanding how black-box models make their predictions and what their limitations are.

Hopefully, the foundation from testing XAI in simulated environments assists in producing knowledge and understanding of how to best make use of these methods in different applications. Gradually, several use cases and data from industry players in the EXAIGON project will be available. This can be used to test the methods more thoroughly in relevant environments in the master's thesis. Several platforms can be used for testing, including robotic arms, drones and marine vessels.

References

[1] OpenAI, *Gym*, https://gym.openai.com/, [Online; accessed 11-October-2020], 2020.

[2] emanual.robotis.com, *Openmanipulator*, http://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/, [Online; accessed 25-August-2020], 2019.

[3] S. Lundberg, *Shap*, https://github.com/slundberg/shap, [Online; accessed 1-September-2020], 2020.

[4] M. Ribeiro, *Lime*, https://github.com/marcotcr/lime, [Online; accessed 20-September-2020], 2020.

[5] Microsoft, *Interpretml - alpha release*, https://github.com/interpretml/interpret, [Online; accessed 25-November-2020], 2020.

[6] S. B. Remman, "Robotic manipulation using deep reinforcement learning," *NTNU Master Thesis*, 2020.

[7] A. e. a. Lekkas, *Exaigon*, https://www.ntnu.edu/exaigon, [Online; accessed 07-December-2020], 2020.

[8] D. Gunning, *Explainable artificial intelligence*, Program Update Darpa, 2017.

[9] A. Adadi and M. Berada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," *IEEE Access*, vol. 6, no. 1, pp. 52 138–52 160, 2018.

[10] P.-J. Kindermans, K. T. Schütt, M. Alber, K.-R. Müller, D. Erhan, B. Kim, and S. Dähne, *Learning how to explain neural networks: Patternnet and patternattribution*, 2017. arXiv: 1705.05598 [stat.ML].

[11] M. Blumreiter, J. Greenyer, F. J. C. Garcia, V. Klös, M. Schwammberger, C. Sommer, A. Vogelsang, and A. Wortmann, *Towards self-explainable cyber-physical systems*, 2019. arXiv: 1908.04698 [cs.AI].

[12] A. Nguyen, J. Yosinski, and J. Clune, *Understanding neural networks via feature visualization: A survey*, 2019. arXiv: 1904.08939 [cs.LG].

[13] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, 2015. arXiv: 1412.6572 [stat.ML].

[14] W. Samek and K.-R. Müller, "Towards explainable artificial intelligence," *Lecture Notes in Computer Science*, pp. 5–22, 2019.

[15] A. Weller, *Transparency: Motivations and challenges*, 2017. arXiv: 1708.01870 [cs.CY].

[16] Z. C. Lipton, *The mythos of model interpretability*, 2016. arXiv: 1606.03490 [cs.LG].

[17] O. Dictionary, *Artificial intelligence*, https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095426960, [Online; accessed 01-October-2020], 2020.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, The MIT Press, 2018.

[19] D. Fumo, *Types of machine learning algorithms you should know*, https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861, [Online; accessed 03-October-2020], 2017.

[20] A. Lekkas, *Lecture notes in ttk23 introduction to autonomous robotics systems for industry 4.0*, https://www.itk.ntnu.no/emner/fordypning/TTK23, October-November 2020.

[21] M. Wiering and M. v. Otterlo, *Reinforcement Learning: State of the art*. Springer-Verlag Berlin Heidelberg, 2012.

[22] Y. Bengio, G. Guyon, V. Dror, G. Lemaire, D. Taylor, and D. Silver, "Deep learning of representations for unsupervised and transfer learning," vol. 7, Jan. 2011.

[23] A. Ng, *Deep learning*, https://on-demand.gputechconf.com/gtc/2015/presentation/S5818-Keynote-Andrew-Ng.pdf, [Online; accessed 10-October-2020], 2015.

[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[25] M. A. Nielsen, *Neural networks and deep learning*, http://neuralnetworksanddeeplearning.com/, [Online; accessed 10-September-2020], 2018.

[26] Claire, *4 simple steps to powerful artificial neural networks in python*, https://www.artificiallyintelligentclaire.com/artificial-neural-networks-python/, [Online; accessed 20-October-2020], 2020.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG].

[28] A. Nayak, *Idea behind lime and shap*, https://towardsdatascience.com/idea-behind-lime-and-shap-b603d35d34eb, [Online; accessed 20-October-2020], 2019.

[29] H. D. Harder, *Model-agnostic methods for interpreting any machine learning model*, https://towardsdatascience.com/model-agnostic-methods-for-interpreting-any-machine-learning-model-4f10787ef504, [Online; accessed 20-October-2020], 2020.

[30] M. T. Ribeiro, S. Singh, and C. Guestrin, *"why should i trust you?": Explaining the predictions of any classifier*, 2016. arXiv: 1602.04938 [cs.LG].

[31] J. Manu, *Interpretability part 3: Opening the black box with lime and shap*, https://www.kdnuggets.com/2019/12/interpretability-part-3-lime-shap.html, [Online; accessed 15-December-2020], 2019.

[32] S. Lundberg and S.-I. Lee, *A unified approach to interpreting model predictions*, 2017. arXiv: 1705.07874 [cs.AI].

[33] R. Cubitt, "The shapley value: Essays in honor of lloyd s. shapley," *The Economic Journal*, vol. 101, no. 406, pp. 644–646, 1991.

[34] S. M. Lundberg and S.-I. Lee, *Consistent feature attribution for tree ensembles*, 2018. arXiv: 1706.06060 [cs.AI].

[35] Unknown, *4 simple steps to powerful artificial neural networks in python*, https://openmole.org/Sensitivity.html, [Online; accessed 01-December-2020], 2020.

[36] M. D. Morris, "Factorial sampling plans for preliminary computational experiments," *Technometrics*, vol. 33, no. 2, pp. 161–174, 1991.

[37] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statisticss*, vol. 29, no. 5, pp. 1189–1232, 2002.

[38] C. Molnar, *Interpretable machine learning - a guide for making black box models explainable*, https://christophm.github.io/interpretable-ml-book/index.html, [Online; accessed 02-December-2020], 2020.

[39] Pytorch, *From research to production*, https://pytorch.org/, [Online; accessed 10-September-2020], 2020.

[40] *Anaconda software distribution*, https://docs.anaconda.com/, version Vers. 2-2.4.0, 2020.

[41] Unknown, *Openai*, https://en.wikipedia.org/wiki/OpenAI, [Online; accessed 11-October-2020], 2020.

[42] prabodhhere, *Solving cartpole-v0 using reinforce*, https://www.kaggle.com/prabodhhere/solving-cartpole-v0-using-reinfor, [Online; accessed 07-October-2020], 2018.

[43] S. Verma, *Train your lunar-lander | reinforcement learning, openai gym*, https://towardsdatascience.com/solving-lunar-lander-openaigym-reinforcement-learning-785675066197, [Online; accessed 09-October-2020], 2019.

[44] S. Thakur, *Lunarlander dqn*, https://github.com/sanketsans/openAIenv/blob/master/DQN/LunarLander/LunarLander_DQN.ipynb, [Online; accessed 10-October-2020], 2020.

[45] S. Lundberg, *Shap documentation*, https://shap.readthedocs.io/en/latest/index.html, [Online; accessed 1-September-2020], 2020.

[46] M. Ribeiro, *Lime documentation*, https://lime-ml.readthedocs.io/en/latest/index.html, [Online; accessed 20-September-2020], 2020.

[47] J. Mak, *Introduction to model interpretability*, http://web.stanford.edu/class/cs224u/materials/cs224u-2020-model-explainability.pdf, [Online; accessed 14-November-2020], 2020.

[48] T. Sigma, *Interpretability methods in machine learning: A brief survey*, https://www.twosigma.com/articles/interpretability-methods-in-machine-learning-a-brief-survey/, [Online; accessed 02-December-2020], 2020.

[49] F. C. Alexandre Heuillet and N. Díaz-Rodríguez, *Explainability in deep reinforcement learning*, http://web.stanford.edu/class/cs224u/materials/cs224u-2020-model-explainability.pdf, [Online; accessed 29-November-2020], 2020.

[50] M. Sundararajan, A. Taly, and Q. Yan, *Axiomatic attribution for deep networks*, https://arxiv.org/pdf/1703.01365.pdf, [Online; accessed 15-November-2020], 2017.

[51] E. Puiutta and E. M. Veith, *Explainable reinforcement learning: A survey*, https://arxiv.org/pdf/2005.06247.pdf, [Online; accessed 20-November-2020], 2020.

[52] G. Liu, O. Schulte, W. Zhu, and Q. Li, *Toward interpretable deep reinforcement learning with linear model u-trees*, https://arxiv.org/pdf/1807.05887.pdf, [Online; accessed 15-November-2020], 2018.