Representation Analysis of Deep Reinforcement Learning algorithms in Robotic Environments

by

Mehran Taghian Jazi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

The rise of Deep Learning (DL) and its assistance in learning complex feature representations significantly impacted Reinforcement Learning (RL). Deep Reinforcement Learning (DRL) made it possible to apply RL to complex real-world problems and even achieve human-level performance. One of these problems is related to robotics. Recently, DRL agents successfully learned optimal behavior in a range of robotic environments. The policy can provide much information from its learned representation. However, this policy is approximated using a neural network and, therefore, is a black box.

Explainable Artificial Intelligence (XAI) is a new AI subfield focusing on interpreting Machine Learning models' behavior. A large part of XAI's literature has emerged on feature relevance techniques to explain a deep neural network (DNN) output processing on images. These techniques have been extended to explain Graph classification tasks using Graph Networks (GN). Nevertheless, these methods haven't been exploited to analyze the DRL agent's behavior learned to perform in a robotic environment.

In this work, we proposed to analyze the representation learned by a DRL agent's policy in a robotic environment. We use graph structure to represent the robot's observation in an entity-relationship manner and graph neural networks as function approximators in DRL. For the interpretation phase, an explainability technique called Layer-wise Relevance Propagation (LRP), a feature relevance technique that had been successfully applied to explain image and graph classification tasks, is used to interpret the learned policy.

We evaluate the information provided by the LRP on two simulated robotic environments on MuJoCo. The experiments and evaluation methods were delicately designed to effectively measure the value of knowledge gained by our approach to analyzing learned representations in the Deep Reinforcement Learning task.

Preface

Part of this thesis has been submitted to ICLR 2023 conference with minor changes in the experiments and Introduction; however, the hypothesis and the method is the same.

To the victims of Flight PS752

Who were flying to their dreams, but ...

There are no incurable diseases – only the lack of will.
There are no worthless herbs – only the lack of knowledge.

– Avicenna, 1037

Acknowledgements

First and foremost, praises and thanks to the Almighty for the showers of blessing and attention throughout my whole life.

I would like to express my deepest gratitude to my supervisor, Prof. Osmar Zaiane, for giving me the opportunity to prove my talents, providing invaluable guidance, encouraging me, and believing in me to be successful in my research path.

I would like to thank Dr. Johannes Gunther for his excellent advice on every part of my research, whose challenging questions shed light on the dark path of my research.

I also want to thank my dearest friend, Sheila Schoepp, for kindly helping and supporting me during some challenging moments of my research.

I would like to thank Mitsubishi Electric Co. for providing financial support for this research. I also want to thank amazing people from Mitsubishi Electric Co., Shotaro Miwa and Yoshihiro Mitsuka, for expressing interest in my work, providing helpful feedback, and supporting me throughout this research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep reinforcement learning has plenty of applications in a variety of fields, including training agents to play Atari games [58], [59], board games [74], [75], complex real-world robotics problems [4], [43], other real-world applications such as resource management in computer clusters [56], network traffic signal control [5], chemical reactions optimization [95], or recommendation systems [93]. Despite the extensive application of DRL in the real world, especially in robotics, it lacks an efficient explainability framework for providing interpretation for the agent's actions. In this work, we decide to fill this hole with the help of Graph Neural Networks (GNNs) and Layer-wise Relevance Propagation (LRP). In the following sections, we first present our hypothesis and then discuss our method and research ideas that form the foundation of our approach.

1.1 Problem and Motivation

In this work, our target is to demonstrate that LRP, which was originally proposed to highlight the most contributing pixels to the image classification, can be used alongside GNNs to identify the contribution of each part of the robot to the decision making by a Deep Reinforcement Learning algorithm. Knowing the contribution of each entity in the robot to the decision-making process is highly important. One application is to provide a **visualization for explaining the training process**, which can be done by identifying the robot's entities contributing to learning a task during the training process.

To get an intuition, assume a child is learning to stand up. During the first stages of learning, they use their hands as assistance; however, in later stages, they can stand up easily without using their hands. Therefore, during the early stages of training, the contribution score of both hands and legs would be high, while during later stages, the contribution of hands drops. Another application is during a malfunction, where part of a robot is broken. Knowing the importance of the broken part helps us **figure out how severe the damage is** and whether the agent can recover from that malfunction or not. This recovery can be in the form of learning a new policy from scratch for the new dynamics or transferring the policy trained in the previous dynamics. If we choose to adapt to the new dynamics after a malfunction, this method can **explain the adaptation process**. To have better intuition, imagine the human's writing task. A right-handed human breaks their right hand; after that, they start using their left hand instead. In the first dynamics, we would say that the contribution of its right hand is the highest in the writing task. In contrast, in the second one, after adaptation to the new dynamics, the importance of its left hand escalates while the right hand's importance drops.

To the best of our knowledge, this is the first work to interpret the policy trained in a robotic environment by identifying each part of the robot's contribution without changing the observation's dimensionality. In the experiments, we try to prove this claim that LRP, along with GNNs, is a practical tool to satisfy this purpose. No prior work has focused on highlighting the contribution of components of a robot to the decision-making in a compact representation of the observation space, in which all the components are important and removing any of them would lead to a drop in performance. Hence, there is no past method to compare our work. This explanation targets expert people dealing with robots.

1.2 Deep Reinforcement Learning

Artificial Intelligence has become an essential part of everyday life. In the past few years, with the emergence of deep learning and high-performance

computing, AI agents' capability and accuracy has significantly developed. One branch of AI which applied deep learning successfully to its algorithms is Reinforcement Learning [30], [31], [51], [59], [68], [69]. Many complex real-world tasks that have been previously done by humans are now possible with the application of Deep Reinforcement Learning (DRL). Recently DRL could achieve a human-level and even better performance in Atari games [59], and is able to be applied successfully to a range of simulated and real-world robotic and control challenging tasks [30], [31]. In order for deep learning, specifically DRL methods, to be applied to the real world, one needs to be able to interpret the agent's behaviour. Otherwise, there would be no trust between the user and the system.

1.3 Explainability in Deep Reinforcement Learning

Explainable Artificial Intelligence (XAI), as an emerging field, has the responsibility to interpret the behaviour of the agent to the end-user [29]. In order to confidently use a system, its behaviour should be transparent and justifiable [64]. In the past few years, there have been plenty of works focusing on transportation [12], [32], healthcare [13], [14], [37], [44], law [10], [38], [78], military [35], [45], cybersecurity, education, entertainment, government [1], and image classification [7], [70], [76], [90], [92], [94]. As a branch of Artificial Intelligence, Reinforcement Learning agents need their behavior to be interpretable. As a branch of XAI, Explainable RL (XRL) has recently emerged to focus on the RL problem specifically. These works aim at interpreting different parts of the RL problem such as reward [42], goal [18], history of interactions [71], [72], and observation and representation analysis [50]. This work aims to analyze representations learned by a DRL algorithm focusing on robotic environments. One method focused on interpreting the agent's representation is State Representation Learning (SRL) [20], [21], [49], [65], [66], [81]. SRL is a feature learning method that learns a low-dimensional representation of the state from high-dimensional raw observations (like pixels of an image)

by capturing the variation in the environment caused by the agent's actions. While SRL methods identify the most relevant features of a high-dimensional observation for learning to act and compact the observation accordingly, we still require highlighting the most relevant features in low-dimensional compact observation space robotic environments. In a compact observation space, all the components are critical to learning the task and removing each would lead to a drop in performance. Our target is to rank these components based on their importance.

1.3.1 Layer-wise Relevance Propagation

Convolutional Neural Networks have been widely applied to images, achieving excellent performance in computer vision tasks. Since these networks train end-to-end on large amounts of data, they form a black box, making it challenging to interpret them. There has been an increasing number of works studying the inner structure and behaviour of CNNs to explain the decisions made by these networks [7], [70], [76], [90], [92], [94]. However, CNNs are designed for grid structured data, such as images, and operate on Euclidean spaces. In many applications, one must deal with data of arbitrary structures, such as graphs. Inspired by the past work in explainability of deep CNNs, Baldassarre et al. [8], and Pope et al. [63] apply these explainability methods to non-Euclidean space of graphs in Graph Neural Networks. Pope et al. [63] analyze the explainability of GNNs using five methods that has been originally applied to CNNs. These 5 methods include gradient-based saliency maps [76], Class Activation Mapping (CAM) [90], Excitation Backpropagation (EB) [90], gradient-weighted CAM (Grad-CAM) [70], and contrastive EB. They evaluate these explainability methods on two graph classification problems: visual scene graphs and molecules. Baldassarre et al. [8] study two main classes of explainability techniques: gradient-based and decomposition-based on a toy dataset and a chemistry task. For the gradient-based methods, they use Sensitivity Analysis [26], and Guided Backpropagation [77]. For the decomposition-based method, they use Layer-wise Relevance Propagation (LRP) [7].

Some work extended the application of saliency methods from classification

to RL, focusing on environments with visual data as states. One example of this application is explaining the DRL agent's behavior in Atari games by visualizing its decisions [28], [40], [41], [86]. Nevertheless, saliency methods in RL have only been applied to RL problems with visual input states.

Our purpose is to identify the participation of each part of a robot in the decision-making process in low-dimensional sensory input robotic environments. LRP, as discussed by Baldassarre et al. [8] has proved to be effective in highlighting the contribution of each part of the graph to the classification task. In order to use LRP in our task to identify contributions, we need to use GNNs as function approximators and use graph representation of the robot in the input. As far as we know, this is the first work focused on applying a saliency method to the decision making in a robotic environment to highlight the importance of each part.

1.4 Graph Neural Networks in DRL

Due to the strong relational inductive bias of the GNNs, Sanchez-Gonzalez et al. [67] propose to apply graph architectures to robots and learnable physics engines. In this work, the robot's bodies are represented using graph nodes and joints using graph edges. During learning, knowledge about body dynamics is encoded in the GNN's node update function, interaction dynamics are encoded in the edge update function, and global system properties are encoded in the global update function. Similar to this work, Wang et al. [84] propose Nervenet to learn a policy structured using graph nets to operate on robots represented as graphs. The difference between this graph representation and the one propose by Sanchez-Gonzalez et al. [67] is that in this work, they use joints as nodes of the graph and physical dependencies between joints as edges. Furthermore, Wang et al. study the transferability and generalizability of their model to new dynamics.

Similar to Wang et al. [84], and Sanchez-Gonzalez et al. [67], we use GNNs as function approximators in the DRL algorithm to learn from graph representations of the robot and apply LRP to the non-Euclidean space of

graphs.

1.5 Graph Representation for Robots

Graphs are used to represent different entities and relationships among them. This flexibility in defining arbitrary shapes of relationships makes them powerful tools for relational reasoning. Inductive bias allows a learning algorithm to prioritize one solution over another (independent of the observed data) in a problem having multiple optimal solutions. Since graph neural networks use graph structure and operations, they possess a strong relational inductive bias that imposes constraints on relationships and interactions among entities [9]. Due to this property, they have a better generalization ability than other types of networks.

In addition to the generalization power of GNNs, one can break the input observation of an RL agent into entities and relations, each having specific features. This entity-relation representation supports a better interpretation of the agent's observation space – in our case, robots. The reason is that not only does it consider the features and state of each component for the decision-making, but it also takes into account the position of that component in the whole structure relative to other entities. Then we can apply the explainability techniques of GNNs to interpret the behaviour of the RL agent using its observation space.

The organization of this work is as follows: In Chapter 3, Graphs and Graph Neural Network structures are discussed in Section 3.1, the details of the robot simulator used for our experiments and how to convert its observation space into graphs are explained in Section 3.2, the Layer-wise Relevance Propagation (LRP) is discussed in Section 3.4, the details of Soft Actor-Critic which is the DRL algorithm used in our problem is described in Section 3.5, and at the end of Chapter 3 we depicted our approach. In Chapter 2 we cover the past work in the XRL, create a taxonomy, and then locate our position among the past work. In Chapter 5 we analyze our approach empirically using two well-known robotic problems. Finally, we discuss our approach's

conclusion and future directions in Chapter 6.

Chapter 2

Related Work

In Chapter 1, we introduced the problem we focus on and the scope of this work. There are plenty of works focusing on providing a reasonable explanation for the Deep Reinforcement Learning tasks. In this chapter, we categorize the past work to locate our problem among similar works.

Reinforcement learning is a complex learning process. In order to successfully learn to operate, many factors can influence the performance. Environment, as a factor, has a number of sub-factors like designing the reward function and how to encode observations. Another factor is the experiences and history of interactions with the environment. The choice of the RL algorithm can be counted as another factor. One branch of RL algorithms develops a model of the environment, which can act as another factor affecting the performance. It even makes it more complicated when combining RL with the representation learning power of Deep Learning (DL) models. Some other factors like policy and value-function networks in policy gradient and actor-critic methods add more complexity to the learning process. In addition, DL models are black-box, and analyzing the representations learned by these models is another challenge [36].

To tackle the challenge of interpreting RL algorithms, recently, many works have been proposed focusing on interpretation using one or more components of the RL algorithm. In the following sections, we cover the category of works based on parts of an RL algorithm they focused on.

2.1 Explanation by Analyzing Components of the Environment

One group of work focuses on the components of the environment such as reward and goal. Juozapaitis et al. [42] propose to decompose the reward function into different meaningful components. Based on the value of those reward types, the agent can explain its behaviour. Wang et al.[83] propose to solve the global reward games in which multiple agents aim to maximize a global reward. Their algorithm can distribute the global reward among multiple agents solving the problem. This distributed reward reflects precisely the contribution of each agent to the global reward.

For goal-oriented interpretation, some work in explainability of RL algorithms with a focus on robotics propose to interpret the agent's decision based on its goal rather than the state's specifications. Cruz et al. [18] is an example that, unlike past work that dedicates its attention to data-driven approaches for the reinforcement learning's explanation, they focus on providing an interpretation of the agent's actions based on its goal. Inspired by the idea of Hindsight Experience Replay (HER) [3], Beyret et al. [11] propose to apply a hierarchical structure to complex multi-step robotic tasks such as Mujoco's `FetchPickAndPlace-v1` robotic environment. They consider a high-level agent, which divides the entire task into smaller ones, and a low-level agent, which is trained to fulfill those smaller tasks. The high-level agent serves as an interpreter between the human, the environment, and the low-level agent controlling the robot's position. Another work that does not have empirical results in robotic environments but can be applied to robotics as well extends HER to the language setting and proposed Textual HER (THER) [16]. In this setting, the agent will receive a textual description of its goal and is rewarded when achieving it. This way, the language generates a level of semantics and interpretability for humans.

2.2 Explanation by Analyzing Experience and History of Interactions

This research category focuses on analyzing the agent's experience and how it interacts with the environment during the learning process. Reinforcement learning agents decide and take actions according to the current situation and do not pay attention to the future or history (Markov property). Moreover, RL agents learn from delayed rewards – the reward received after executing some action should be "propagated" back to the states and actions leading to that situation. These two situations make it hard to explain the behaviour of an RL agent. Sequeira et al. [71], [72] propose a framework that uses introspection analysis of an agent's history of interactions with the environment to extract interestingness elements regarding its behaviour. Then, an explanation framework uses these interesting elements to expose the agent's behaviour to a human user. Dao et al. [19] propose *snapshot images*, a monitoring model to record the most important moments from experience. With this, one can diagnose the most influential transition tuples for a policy's individual decisions. Gottesman et al. [27] identify the observations in the data whose removal will significantly affect the estimation of Q-values, thus highlighting the important experiences.

One subcategory of this line of work focuses on influential trajectories rather than important individual transition tuples. Amir and Amir [2] develop a method to produce a summary of an agent's behavior by extracting important trajectories from simulations of the agent and, therefore, help choose between agents or determine the level of autonomy the agent can operate. Huang et al. [39] propose to diagnose the critical states in which it is crucial to take a certain action. To identify these states, they select states where the chosen action has a much higher Q-value than another. Lage et al.[46] explore the effect of using different policy summarization methods on the ability to reconstruct a policy. These policy summarization methods are used to extract subsets of state-action pairs that best characterize the agent's behavior.

2.3 Explanation via Behavior Prediction

Some work wants to explain the decisions made by an RL agent by knowing what the expected behavior of an agent is based on current and history of interactions. Cruz et al. [17] propose a memory-based explainable RL, using which the agent can explain its decisions in terms of the probability of success and the number of transitions to reach the goal. In a later work by Cruz et al. [18] they add two other approaches to the memory-based method: learning-based and introspection-based. These methods are different in terms of space complexity, where the two latter methods have a reduced space complexity making them suitable for domains requiring a continuous state representation. Lin et al. [53] propose an embedded self-prediction model to learn action-values directly represented via human-understandable properties of expected futures. The authors claim that by contrasting these properties predicted for each action, the action preferences could be explained. Yau et al. [87] propose a method to obtain a projection of predicted future trajectories from a current observation and propose action. In other words, this method explains what outcomes are expected by RL agents.

2.4 Explanation by Developing an Augmented Model

In this line of research, a model is trained simultaneously with the agent to explain its behavior. Chen et al. [15] put forward an interpretable deep RL method for end-to-end autonomous driving. This method employs a latent space (with a sequential latent environment model) to encode a complex urban driving environment into which historical high-dimensional raw observations are compressed. This model is learned jointly with the maximum entropy RL process. Madumal et al. [55], inspired by causal relationships, introduce an action influence model for model-free RL agents, which approximates the causal model of the environment relative to the actions taken by the agent. Their approach learns a structural causal model during reinforcement learning

and encodes causal relationships between variables of interest. Then it generates explanations for why and why not questions by counterfactual analysis. Volodin et al. [82] define the simplicity of causal explanations via the sparsity of the causal model that describes the environment. They propose a framework containing a learned mapping from observations to latent features—a model predicting latent features at the next time steps given ones from the current time-step. A sparse causal graph is trained jointly with the RL agent.

2.5 Explanation by Training a Transparent Policy

Transparent algorithms in machine learning are known to be explainable by themselves, e.g., decision trees (DTs) and rule-based methods. Based on this idea, some authors present methods for training an inherently interpretable policy using decision trees and other transparent algorithms. Since DTs are not differentiable, Silva et al. [73] propose to learn a soft DT, in which sigmoid activation functions replace the boolean decisions in classic DTs. Liu et al. [54] apply mimic learning to make a trade-off between the performance and interpretability of the DRL model. In order to make the DRL neural net interpretable, they propose Linear Model U-Tree (LMUT), a version of U-tree which contains a linear model at each leaf node to strengthen the generalization ability. Topin et al. [80] introduce CUSTARD to maintain the interpretability advantage of a DT policy while using a non-interpretable neural network for training. Additionally, They present a new MDP representation for learning a DT policy for a base MDP. Other methods providing transparent policies include the usage of symbolic expressions [47], basic algebraic equations [34], and logic expressions [91] to provide an inherently interpretable policy.

2.6 Explanation through Representation Analysis

One group of work propose techniques to analyze the representations learned by a policy using which the decisions made by an RL agent could be explained. Garnelo et al.[25] combine the representation power of deep learning models with symbolic AI to learn a representation that is easily comprehensible to humans. Another point of view in providing interpretable representation is considering Relational RL, which advocates the use of relational state, action, and policy representation. Zambaldi et al. [89] propose a method that uses a self-attention mechanism to iteratively reason about the relations between entities in a scene. To do that, they combine RL with Inductive Logic Programming by representing states, actions, and policies using a first-order (relational) language. Another sub-group of this kind focuses on State Representation Learning (SRL) which aims at learning compact representations from raw observations that help speed up policy learning, make it easier to interpret the learned policy, and improve performance [49]. SRL is especially useful in RL for robotic and control, which helps agents learn an interpretable policy by reducing the high-dimensional observation, allowing them to analyze the representations learned by the RL agent [50]. The SRL has been applied to improve the performance in robotic tasks [20], [21], [65], [66], [81]. By applying SRL techniques, an interpretable representation of the observations learned by the agent can be provided. However, there is a trade-off between the performance and interpretability of the model.

While the works mentioned above try to learn an interpretable representation by the RL agent that is easily understandable by humans, most of them cannot be generalized to other tasks because of the trade-off between learning an interpretable representation and performance. Therefore, separating the explanation phase from the learning phase is crucial. *Post hoc* methods in explainability are techniques applied to models after the learning phase has finished extracting useful interpretable information from the learned model. Some works use natural language to provide explanations for an agent's be-

havior [22], [33], [85]. However, these explanations are task-specific. Another approach to post hoc analyzing the learned representation is the work by Zahavi et al. [88] in which they explore the features extracted by the DQN algorithm. They discuss that features learned by DQN belong to different clusters, among which they identify hierarchical structures. Using this, they could explain the successful performance of DQN in Atari games. One group of powerful methods to analyze learned representations that have proven their efficiency in explaining image classification tasks is saliency methods. Recently in deep RL, some works apply this approach to interpreting the agent's decision based on its state, where the states are represented using images. Weitkamp et al. [86], Greydanus et al. [28], and Iyer et al. [41] focus on providing explainability for the behaviour of DRL agents on Atari games and visualize the decision process using saliency map methods. Huber et al. [40] which apply Layer-wise Relevance Propagation (LRP) to create a saliency map of the most relevant pixels in the states of an Atari game used by a dueling DQN algorithm to generate actions. Their focus is on explaining the Atari games environment. A complete list of saliency methods applied to RL can be found in the work by Atrey et al. [6].

Our work locates in the last category, which means it focuses on analyzing representations learned by the policy in a policy gradient algorithm. Specifically, we focus on robotic environments, extracting the necessary information from the learned policy in a robotic environment to find the most contributing components of the robot to both observing and acting phases. Our method is different from the SRL technique, which learns a low-dimensional representation of the state from high-dimensional raw observations (like pixels of an image) by capturing the variation in the environment caused by the agent's actions. While SRL methods identify the most relevant features of a high-dimensional observation for learning to act and compact the observation accordingly, in our work, we propose to highlight the most relevant features in low-dimensional compact observation space robotic environments. In a compact observation space, all the components are critical to learning the task and removing each would lead to a drop in performance. Our target is to rank

these components based on their importance.

In order to decompose the robot into its components and analyze the contribution of each one separately, we use graphs as a representation method for observations. Then, inspired by the explainability techniques proposed for graph classification tasks [8], [63], we proceed to analyze representations learned by a DRL agent in a robotic environment. We apply the most efficient technique in the graph classification task, as discussed by Baldassarre et al. [8] to our work. This technique is Layer-wise Relevance Propagation [7]. Our approach is discussed in Chapter 4.

Chapter 3

Background

The general approach to analyze the representations learned by a Deep RL algorithm is inspired by the explainability of graph neural networks in a graph classification task. One method that was successfully applied to interpret the graph classification was Layer-wise Relevance propagation [8]. The LRP is a decomposition-based method, originally proposed to explain image classification by decomposing the output probability given to a specific class by the classifier and back-propagate that probability to the input image. On the other hand, a robot's structure is a graph, connecting nodes (limbs) to each other using edges (joints). Combining the idea of explainablity of graph neural networks using LRP, and similarity of robots' structures to graphs, our method aims to analyze the representations learned by a policy in a Deep RL algorithm. The structure of this chapter is as follows: the details of Graph Neural Network's operations are explained in section 3.1, the conversion of a robot's observation space to a graph of observation is introduced in 3.2, the layer-wise relevance propagation is explained in 3.4, the Deep RL algorithm used in our problem is described in 3.5, and the general framework for representation analysis is provided in 4.1.

3.1 Graphs and Graph Neural Networks

3.1.1 Graphs

A *graph* is a structure made of vertices that are connected through edges. In computer science, a graph is used to represent structured entities with relations between each pair. Each node of the graph can be considered to be an *entity*. The *relations* between those entities can be shown using an edge between a pair of nodes (entities). Therefore, a graph is used to represent those kinds of data that are able to be expressed in an entity-relationship manner.

3.1.2 Relational Inductive Biases

As discussed by Battaglia et al. [9] *relational reasoning* involves manipulating structured representations of *entities* and *relations*, using *rules* for how they can be composed. An *entity* is an element with attributes, such as Atoms in a molecule that have mass and specific atomic properties related to their nuclear structure and electron configuration. A *relation* is a property between entities. Some relations have attributes as well. In the molecule example, different atoms connect using various chemical bonds depending on their properties. Therefore, each bond (relation) can have specific attributes. Sometimes, the form of relations between entities can affect the global context, such as a molecule's properties that depend upon its structure (bonds between atoms). A *rule* is a function (like a non-binary logical predicate) that maps entities and relations to other entities and relations. An example of a unary rule is like "*is entity X large*" or binary like "*is entity X larger than entity Y*".

Learning is the process of finding a solution that best explains the current state of the world. In many cases, there are multiple good solutions. In this situation, the *inductive bias* of a learning algorithm allows selecting one solution among different options, independent of the observed data [57]. For machine learning approaches with a capacity for relational reasoning, Battaglia et al. [9] introduced *relational inductive bias*, which refers to inductive biases that impose constraints on relationships and interactions among entities in a learning process.

3.1.3 Graph Neural Networks and Relational inductive bias

In deep learning, there are different relational inductive biases based on type of the neural network architecture. For each specific type of neural network, we must specify *entities*, *relations*, and *rules* for composing entities and relations. For example, in a *fully connected layer* building block of a neural network, if we consider layer i with N neurons and layer j with M neurons, each neuron $x_{(n,i)}$ (unit n in layer i) and $x_{(m,j)}$ (unit m in layer j) is considered *entities* of the network. The *relations* between entities are all-to-all (all units in layer i are connected to all units in layer j). The *rules* are specified as follows:

$$x_{(m,j)} = \phi\left(\sum_{n=1}^{N} x_{(n,i)}.w_{(n,m)} + b_m\right)$$

where $w_{(n,m)}$ is the weight between neuron n of layer i and neuron m of layer j, b_m is the bias term for neuron m of layer j, and ϕ is a non-linearity such as a rectified linear unit (ReLU). Since all units in layer i interact to determine the units in layer j, the implicit relational inductive bias is weak in a fully connected layer.

The same thing is true for the Convolutional Neural Network [24][48], except that the weights are in the form of a kernel being convolved to some part of the input. Due to this process, each output unit involves convolving some part of the input with a kernel, adding a bias term and applying a non-linearity. The convolving process imposes two types of relational inductive bias. First, entities in close proximity interact to produce the output unit (consider pixels of an input image inside a kernel), called *locality*. Second, the same kernel is being convolved to different parts of the input, causing *translation invariance*. Therefore, convolutional layers have some spatial relational inductive bias.

For the recurrent layer [23], the entities are inputs and hidden states at each processing step. The relations are the dependence of the current hidden state on the previous hidden state and current input. The rule for composing entities and relations takes input at time-step t and the previous hidden state to update the current hidden state. This rule is reused across steps, reflecting

a relational inductive bias of temporal invariance (the rule is similar across time steps, similar to CNN in which kernel is similar spatially).

While these deep learning models contain some relational inductive biases, we seek deep learning building blocks for representing various kinds of entity relationships and their rules. In other words, we want to specify the relational inductive bias of the deep learning architecture. Graph Neural Network is such a tool to satisfy this requirement. Using this tool, we can explicitly represent entities and their relations, along with learning algorithms that find rules for computing the interactions. Therefore, GNNs enjoy a solid relational inductive bias beyond what is offered by CNNs and RNNs.

3.1.4 Computation in Graph Neural Networks

The internal structure of a GNN and the computation steps are adapted from the work by Battaglia et al. [9].

Internal structure of a GN block

The main unit of computation in a GN is a GN block, which takes as input a graph, performs computation over the structure, and returns a graph as output. Entities in a GN are represented as graph's *nodes* and relations as graph's *edges* and system-level properties as *global* attributes. Each graph is defined as a 3-tuple $G = (u, V, E)$, where u is the global attribute; $V = \{\mathbf{v_i}\}_{i=1}^{N^v}$ is the set of vertices where N^v is the number of nodes and $\mathbf{v_i}$ is a node's attribute; $E = \{(\mathbf{e_k}, r_k, s_k)\}_{k=1}^{N^e}$ is the set of edges where N^e is the number of edges, r_k is the index of the receiver node, s_k is the index of the sender node, and $\mathbf{e_k}$ is an edge's attribute.

The update and aggregate functions within a GN block, denoted by ϕ and ρ respectively, are as follows:

$$\begin{aligned} \mathbf{e}'_k &= \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) & \overline{\mathbf{e}}'_i &= \rho^{e\to v}(E'_i) \\ \mathbf{v}'_i &= \phi^v(\overline{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) & \overline{\mathbf{e}}' &= \rho^{e\to u}(E') \\ \mathbf{u}' &= \phi^u(\overline{\mathbf{e}}', \overline{\mathbf{v}}', \mathbf{u}) & \overline{\mathbf{v}}' &= \rho^{v\to u}(V') \end{aligned} \tag{3.1}$$

where $E'_i = \{(\mathbf{e'_k}, r_k, s_k)\}_{r_k=i, k=1:N^e}$ is the updated set of edges whose receiver is

node i, $V' = \{\mathbf{v_i'}\}_{i=1}^{N^v}$ is the set of updated nodes, $E' = \bigcup_i E_i' = \{(\mathbf{e_k'}, r_k, s_k)\}_{k=1}^{N^e}$ is the set of all the updated edges. The ϕ^e is an update function that updates the edge's attribute given the edge's previous attribute, receiver node, sender node, and global system attributes. We can specify whether it updates the edge's attribute by considering sender, receiver, or global attributes or non of them. The ϕ^v is an update function which updates node i's attribute given node i's previous attribute, aggregation of the updated edge features whose receiver node is node i, and current global attributes of the system. Again one can specify whether to consider edge or global attributes or only consider the current node's features to do the update. ϕ^u operates in the same way as ϕ^e and ϕ^v in that, given the aggregation of the updated node and edge attributes, along with current global features, it updates the global attributes. The ρ functions are aggregated functions that take a set as input and reduce it to a single element representing the aggregated information. $\rho^{e\to v}$ aggregates edge attributes of the edges whose receiver node is the same, $\rho^{e\to u}$ aggregates edge attributes of all the edges, $\rho^{v\to u}$ aggregates node attributes of all the nodes. These aggregate functions can be either a sum-, mean-, or max-pooling over the set of inputs.

Computation steps in GNNs

The computation steps of a GN block is summarized in algorithm 1. These steps comprise of 3 main updates:

1. The first part relates to edge updates. The algorithm updates edge attributes in lines 2-4 using the ϕ^e function.

2. The second part is dedicated to updating node attributes (lines 5-9). In this part, first, we select the set of edges whose receiver node is node i denoted by E_i' (line 6). Then the edge features of the edges in E_i' would be aggregated using $\rho^{e\to v}$ to be used in the update function ϕ^v to update attributes of node i in line 8.

3. The last part is updating global attributes (lines 10-14). V' and E' denote sets of updated node and edge attributes, respectively. Firstly,

the updated edge attributes are aggregated using $\rho^{e\to u}$ and called $\overline{\mathbf{e}}'$. Secondly, the updated node attributes are aggregated using $\rho^{v\to u}$ and called $\overline{\mathbf{v}}'$. Then these aggregated node and edge attributes are used in updating global attributes using ϕ^u.

The order of steps in algorithm 1 is irrelevant. One can change this order to update global attributes, per-node attributes, then per-edge attributes.

Algorithm 1 Steps of computation in a GN block

function GRAPH NEURAL NETWORK($E, V, \mathbf{u}$)
  for $k \in \{1 \dots N^e\}$ **do**
    $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$ ▷ Compute updated edge attributes
  end for
  for $i \in \{1 \dots N^v\}$ **do**
    let $E'_i = \{(\mathbf{e}'_\mathbf{k}, r_k, s_k)\}_{r_k=i, k=1:N^e}$
    $\overline{\mathbf{e}}'_i = \rho^{e\to v}(E'_i)$ ▷ Aggregate edge attributes per node
    $\mathbf{v}'_i = \phi^v(\overline{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$ ▷ Compute updated node attributes
  end for
  let $V' = \{\mathbf{v}'_i\}_{i=1}^{N^v}$
  let $E' = \{(\mathbf{e}'_\mathbf{k}, r_k, s_k)\}_{k=1}^{N^e}$
  $\overline{\mathbf{e}}' = \rho^{e\to u}(E')$ ▷ Aggregate edge attributes globally
  $\overline{\mathbf{v}}' = \rho^{v\to u}(V')$ ▷ Aggregate node attributes globally
  $\mathbf{u}' = \phi^u(\overline{\mathbf{e}}', \overline{\mathbf{v}}', \mathbf{u})$ ▷ Compute updated global attribute
  return $(E', V', \mathbf{u}')$
end function

3.1.5 Motivation to our Problem

A robot's structure is composed of different parts having specific attributes. The communication of all these parts leads to the movement of the robot to pursue a specific goal. Therefore, each part of the robot contributes to the decision-making process. In this problem, the purpose is to analyze the contribution of each part as an element of a system. Therefore, we require to break the robot to separate components that communicate with each other. To satisfy this requirement, we make use of graphs. In order to learn how to make decisions given the robot's observation graph at each time step, we apply Deep Reinforcement Learning algorithms with GNNs. The following sections will discuss the details of decomposing a robot to separate components and

converting it to a graph, how to use DRL algorithms with graphs, and how graphs can help explain an agent's decisions.

3.2 Graphs in Robotics

In the previous chapter, we discussed graph neural networks, operations inside a GN block, and the goal of using graphs in our problem. One primary step in providing explanation for a robot is to decompose its structure into separate components. To satisfy this requirement, we take advantage of graphs and their properties. This chapter explains the steps of converting a robot's vector of observation into a graph. Specifically, we used robots in the OpenAI gym simulator to run experiments. Therefore, all the methods here are described according to the OpenAI gym API but can be extended to other robotic environments.

3.2.1 Mujoco Physics Engine

Mujoco [79] is a widely used simulation environment and a standard benchmark for testing reinforcement learning strategies and algorithms. Mujoco is a C/C++ library with a C API. **OpenAi gym** is a toolkit that exerted the Mujoco library to generate a python interface for application in research. For our experiments, we are going to use 2 different environments from OpenAI gym, namely `HalfCheetah-v2` and `Walker2D-v2` . The following sections will explain the structure of the OpenAI gym environments' files. Then the strategy to change the original observation space of the OpenAI gym environments to a graph is discussed.

3.2.2 OpenAI gym structure

For each environment, the files are categorized into two groups: one for defining the static structure and another for the dynamic behaviour of the robot.

The static structure of a robot is stored in a XML format that includes joints, links, and how they are attached to form the robot. In addition, there are some specific attributes related to each part of the robot. The XML tags are used to represent the robot's components. The XML attributes are used to represent the features of a specific limb. The only parts (XML tags) that we are dealing with are `<body>` and `<joint>` tags. We will discuss further how

we use these tags and their attributes to manipulate the original observation space to a graph of observation in section 3.2.3.

Each environment also has python files that define its dynamic behavior at run-time. This file should at least contain `reset()` and `step(action)` functions. The process of the agent's interaction with the environment starts by calling `reset()` function, which returns an initial observation $s_0 \in S$. Then at each time step t, the predicted action a_t based on the current observation s_t would be given to the `step` function as `step(`a_t`)`. This `step` function returns 4 different values [60]:

- *Observation* (s_{t+1}): an environment-specific object representing the observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game.

- *Reward* (`float`): amount of reward achieved by the previous action. The scale varies between environments, but the goal is to increase the total reward.

- *Done* (`boolean`): whether it's time to `reset` the environment again. Most (but not all) tasks are divided up into well-defined episodes, and done being True indicates the episode has terminated.

- `Info` (`dict`): diagnostic information useful for debugging. It can sometimes be useful for learning (for example, it might contain the raw probabilities behind the environment's last state change). However, official evaluations of your agent are not allowed to use this for learning.

3.2.3 Change observation to Graph

The general idea of shifting from a vector of observation to a graph of observation is similar to the work by Sanchez-Gonzalez et al. [67] with some minor differences. Sanchez-Gonzalez et al. considered each `body` to be a node and each `joint` to be an edge that connects body parts. In our work, we treat `body`s in the same way; however, the edges are either moving `joint`s or welded.

The robot's graph representation forms a tree with no cycles; otherwise, the robot would not be able to move its edges (joints).

Kinematic tree

The main model of the robot is an XML tree created by nested `body` elements. The top level body is special and is called **worldbody**. When a body `<body name="body1">` is connected to another body `<body name="body2">` in the robot, then `<body name="body2">` would be a child element of the parent body `<body name="body1">` in the sense of XML.

When a `joint` is defined inside a body, its function is to create motion degrees of freedom between them – e.g. figure 3.1. If no joints are defined within a given body, that body is welded to its parent – e.g. figure 3.2.

```
<body name="body_1" pos="0.2 0.2 0">
    <body name="body_2" pos="0.2 0.2 0" >
        <joint axis="-1 1 0" name="ankle" pos="0.0 0.0 0.0" range="30 70" type="hinge"/>
    </body>
</body>
```

Figure 3.1: A joint connection defined in the robot's kinematic tree. In this connection, the child body "body_2" is connected to the parent body "body_1" through "ankle" joint.

```
<body name="camera_base" pos="0 0.02 0">
    <inertial diaginertia="0 0 0" mass="0" pos="0 0.02 0"/>
    <body name="camera_body" pos="0 0 0" quat="0.5 -0.5 0.5 -0.5">
        <inertial diaginertia="0 0 0" mass="0" pos="0 0 0" quat="0.5 -0.5 0.5 -0.5"/>
        <camera euler="3.1415 0 0" fovy="50" name="head_camera_rgb" pos="0 0 0"/>
    </body>
</body>
```

Figure 3.2: A welded connection in the robot's kinematic tree. In this connection, the "camera_body" contains a camera and is welded to the "camera_base" body.

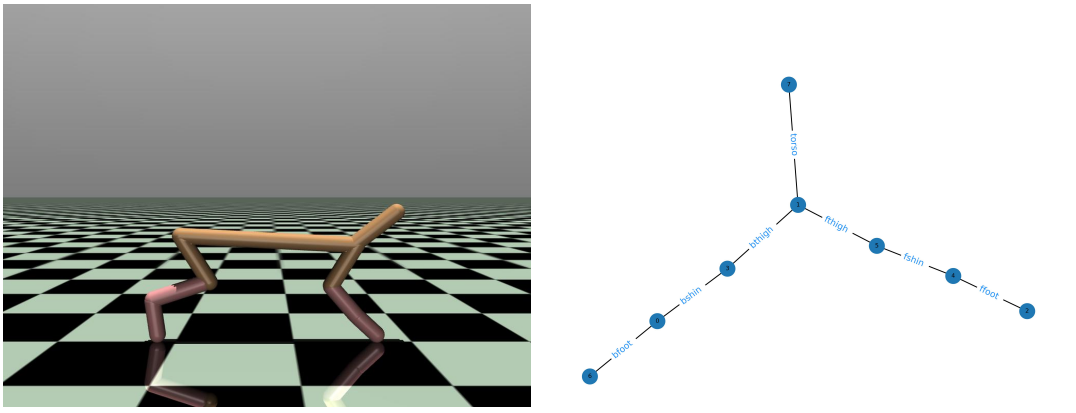
Other elements can be defined within the tree created by nested body elements, in particular `geom`, `site`, `camera`, `light`. We do not consider these elements for our graph of observation. When an element is defined within a body, it is fixed to the body's local frame and always moves with it. Elements

that refer to multiple bodies or do not refer to bodies at all, are defined in separate sections outside the kinematic tree [52].

Each node and edge in the observation graph has a feature vector. Since we have two types of edges, we need to specify feature vectors specific to each type. For the welded edges, the feature vector is all zeros. For joints, the feature vector contains information about the joint's positions (joint's `qpos`) and joint's velocities (joint's `qvel`). We have no features selected for nodes because only edges matter to create movements. Although nodes of the graph have no features, we still take advantage of them in graph operations.

3.3 Environments

3.3.1 HalfCheetah-v2



(a) HalfCheetah-v2 robot in MuJoCo simulator.

(b) Graph representation of the HalfCheetah-v2 robot

Figure 3.3: `HalfCheetah-v2` robot and graph

The HalfCheetah is a 2 dimensional robot, consisting of 9 links and 8 joints (as depicted in Figure 3.3a). In this environment, the goal is to apply actions (torques) to the group of joints to make the cheetah run forward as fast as possible. The reward is positive for moving forward and negative for moving backward. The cheetah's torso and head are fixed, and the torques can only be applied on the other 6 remaining joints: front and back thighs (connecting to the torso), shins (connecting to the thighs) and feet (connecting to the

shins). All the details of the `HalfCheetah-v2` environment are acquired from the Gym documentation [61].

- **Action Space** The action space is a vector of length 6, and each element can have values of type `float32` in range $[-1.0, 1.0]$. Table 3.1 reflects the details of each action element.

- **Observation Space** Observations consist of positional values of different body parts of the cheetah, followed by the velocities of those individual parts (their derivatives) with all the positions ordered before all the velocities. In the documentation, it mentions that the x-coordinate of the center of mass is excluded from the observation space. However, we added that element to our observation space, which is part of the `torso`. Therefore, our observation is a `ndarray` with shape $(18,)$. The details of the observation space are reflected in table 3.2

- **Reward** The reward consists of two parts:

  - $forward\_reward$: A reward of moving forward which is measured as follows:

    $$\begin{aligned} forward\_reward = & forward\_reward\_weight \times \\ & (x\_coordinate(\text{before action}) - \\ & x\_coordinate(\text{after action}))\,/dt \end{aligned}$$

    where dt is the time between actions and is dependent on the $frame\_skip$ parameter (fixed to 5), where the frametime is 0.01 – making the default $dt = 5 * 0.01 = 0.05$. This reward would be positive if the cheetah runs forward (right).

  - $ctrl\_cost$: A cost for penalising the cheetah if it takes actions that are too large. It is measured as $ctrl\_cost\_weight \times sum(action^2)$ where $ctrl\_cost\_weight$ is a parameter set for the control and has a default value of 0.1.

  The total reward returned is $reward = forward\_reward - ctrl\_cost$ and info will also contain the individual reward terms

The XML model of the `HalfCheetah-v2` is in Section B.1.

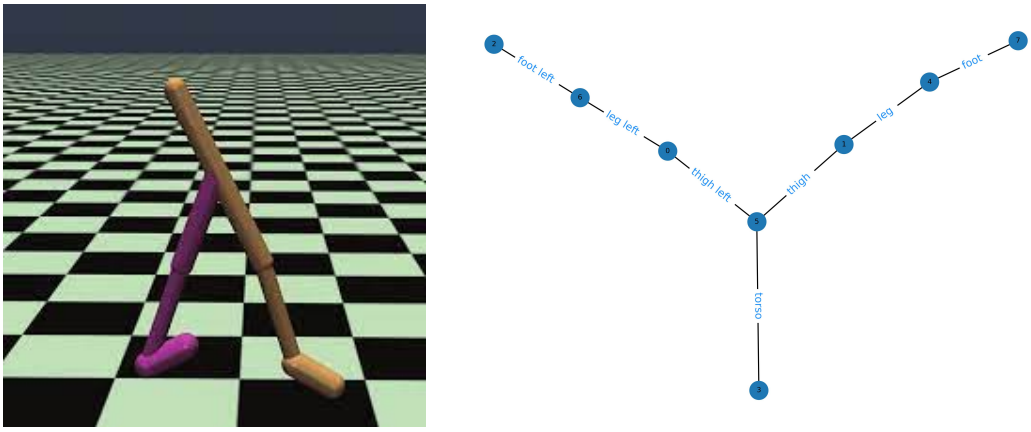
Table 3.1: Details of the action space in the `HalfCheetah-v2` environment.

Num	Action	Control Min	Control Max	Name (In corresponding XML file)	Joint Type	Unit
0	Torque applied on the back thigh rotor	-1	1	bthigh	hinge	torque (Nm)
1	Torque applied on the back shin rotor	-1	1	bshin	hinge	torque (Nm)
2	Torque applied on the back foot rotor	-1	1	bfoot	hinge	torque (Nm)
3	Torque applied on the front thigh rotor	-1	1	fthigh	hinge	torque (Nm)
4	Torque applied on the front shin rotor	-1	1	fshin	hinge	torque (Nm)
5	Torque applied on the front foot rotor	-1	1	ffoot	hinge	torque (Nm)

Table 3.2: Details of the observation space in the `HalfCheetah-v2` environment.

Num	Observation	Min	Max	Name (In corresponding XML file)	Joint Type	Unit
0	z-coordinate of the front tip	$-\infty$	$+\infty$	rootz	slide	position (m)
1	angle of the front tip	$-\infty$	$+\infty$	rooty	hinge	angle (rad)
2	angle of the second rotor	$-\infty$	$+\infty$	bthigh	hinge	angle (rad)
3	angle of the second rotor	$-\infty$	$+\infty$	bshin	hinge	angle (rad)
4	velocity of the tip along the x-axis	$-\infty$	$+\infty$	bfoot	hinge	angle (rad)
5	velocity of the tip along the y-axis	$-\infty$	$+\infty$	fthigh	hinge	angle (rad)
6	angular velocity of the front tip	$-\infty$	$+\infty$	fshin	hinge	angle (rad)
7	angular velocity of the second rotor	$-\infty$	$+\infty$	ffoot	hinge	angle (rad)
8	x-coordinate of the front tip	$-\infty$	$+\infty$	rootx	slide	velocity (m/s)
9	y-coordinate of the front tip	$-\infty$	$+\infty$	rootz	slide	velocity (m/s)
10	angle of the front tip	$-\infty$	$+\infty$	rooty	hinge	angular velocity (rad/s)
11	angle of the second rotor	$-\infty$	$+\infty$	bthigh	hinge	angular velocity (rad/s)
12	angle of the second rotor	$-\infty$	$+\infty$	bshin	hinge	angular velocity (rad/s)
13	velocity of the tip along the x-axis	$-\infty$	$+\infty$	bfoot	hinge	angular velocity (rad/s)
14	velocity of the tip along the y-axis	$-\infty$	$+\infty$	fthigh	hinge	angular velocity (rad/s)
15	angular velocity of the front tip	$-\infty$	$+\infty$	fshin	hinge	angular velocity (rad/s)
16	angular velocity of the second rotor	$-\infty$	$+\infty$	ffoot	hinge	angular velocity (rad/s)

3.3.2 Walker2D-v2



(a) Walker2d-v2 robot in MuJoCo simulator.

(b) Graph representation of the Walker2d-v2 robot

Figure 3.4: `Walker2D-v2` robot and graph

The walker is a two-dimensional two-legged figure that consists of four main body parts (as depicted in Figure 3.4a) – a single torso at the top (with the two legs splitting after the torso), two thighs in the middle below the torso, two legs in the bottom below the thighs, and two feet attached to the legs on which the entire body rests. The goal is to make coordinate both sets of feet, legs, and thighs to move in the forward (right) direction by applying torques on the six hinges connecting the six body parts [62]. Now we will cover the details of each components of the environment according to the OpenAI Gym documentation [62].

- **Action Space** The action space is a vector of length 6, and each element can have values of type `float32` in range $[-1.0, 1.0]$. Table 3.3 reflects the details of each action element.

- **Observation Space** Observations consist of positional values of different body parts of the walker, followed by the velocities of those individual parts (their derivatives) with all the positions ordered before all the velocities. In the documentation, it mentions that the x-coordinate of the top is excluded from the observation space. However, we added that

element to our observation space, which is part of the `torso`. Therefore, our observation is a `ndarray` with shape $(18,)$. The details of the observation space are reflected in table 3.4.

- **Reward** The reward consists of three parts:
  - *healthy_reward*: Every timestep that the walker is alive, it receives a fixed reward of value *healthy_reward*,
  - $forward\_reward$: A reward of walking forward which is measured as follows:

$$\begin{aligned} forward\_reward = & forward\_reward\_weight \times \\ & (x\_coordinate(\text{before action}) - \\ & x\_coordinate(\text{after action}))\,/dt \end{aligned}$$

    where dt is the time between actions and is dependent on the $frame\_skip$ parameter (default is 4), where the frametime is 0.002 - making the default $dt = 4 * 0.002 = 0.008$. This reward would be positive if the walker walks forward (right) desired.
  - *ctrl_cost*: A cost for penalising the walker if it takes actions that are too large. It is measured as $ctrl\_cost\_weight * sum(action^2)$ where *ctrl_cost_weight* is a parameter set for the control and has a default value of 0.001.

  The total reward returned is $reward = healthy\_reward + forward\_reward - ctrl\_cost$.

The XML model of the `Walker2D-v2` is in Section B.2.

Table 3.3: Details of the action space in the `Walker2D-v2` environment.

Num	Action	Control Min	Control Max	Name (In corresponding XML file)	Joint Type	Unit
0	Torque applied on the thigh rotor	-1	1	thigh_joint	hinge	torque (Nm)
1	Torque applied on the leg rotor	-1	1	leg_joint	hinge	torque (Nm)
2	Torque applied on the foot rotor	-1	1	foot_joint	hinge	torque (Nm)
3	Torque applied on the left thigh rotor	-1	1	thigh_left_joint	hinge	torque (Nm)
4	Torque applied on the left leg rotor	-1	1	leg_left_joint	hinge	torque (Nm)
5	Torque applied on the left foot rotor	-1	1	foot_left_joint	hinge	torque (Nm)

Table 3.4: Details of the observation space in the `Walker2D-v2` environment.

Num	Observation	Min	Max	Name (In corresponding XML file)	Joint Type	Unit
0	z-coordinate of the top (height of walker)	$-\infty$	$+\infty$	rootz (torso)	slide	position (m)
1	angle of the top	$-\infty$	$+\infty$	rooty (torso)	hinge	angle (rad)
2	angle of the thigh joint	$-\infty$	$+\infty$	thigh_joint	hinge	angle (rad)
3	angle of the leg joint	$-\infty$	$+\infty$	leg_joint	hinge	angle (rad)
4	angle of the foot joint	$-\infty$	$+\infty$	foot_joint	hinge	angle (rad)
5	angle of the left thigh joint	$-\infty$	$+\infty$	thigh_left_joint	hinge	angle (rad)
6	angle of the left leg joint	$-\infty$	$+\infty$	leg_left_joint	hinge	angle (rad)
7	angle of the left foot joint	$-\infty$	$+\infty$	foot_left_joint	hinge	angle (rad)
8	velocity of the x-coordinate of the top	$-\infty$	$+\infty$	rootx	slide	velocity (m/s)
9	velocity of the z-coordinate (height) of the top	$-\infty$	$+\infty$	rootz	slide	velocity (m/s)
10	angular velocity of the angle of the top	$-\infty$	$+\infty$	rooty	hinge	angular velocity (rad/s)
11	angular velocity of the thigh hinge	$-\infty$	$+\infty$	thigh_joint	hinge	angular velocity (rad/s)
12	angular velocity of the leg hinge	$-\infty$	$+\infty$	leg_joint	hinge	angular velocity (rad/s)
13	angular velocity of the foot hinge	$-\infty$	$+\infty$	foot_joint	hinge	angular velocity (rad/s)
14	angular velocity of the left thigh hinge	$-\infty$	$+\infty$	thigh_left_joint	hinge	angular velocity (rad/s)
15	angular velocity of the left leg hinge	$-\infty$	$+\infty$	leg_left_joint	hinge	angular velocity (rad/s)
16	angular velocity of the left foot hinge	$-\infty$	$+\infty$	foot_left_joint	hinge	angular velocity (rad/s)

3.4 Layer-wise Relevance Propagation

Layer-wise Relevance Propagation (LRP) is a saliency map method originally proposed to provide interpretation for the image classification task [7]. In the case of images, the idea of LRP is based on pixel-wise decomposition. The purpose of pixel-wise decomposition is to understand the contribution of a single-pixel of an image x to the prediction $f(x)$ made by a classifier f in an image classification task [7]. However, the purpose of our work has nothing to do with images and pixels; rather, it focuses on graphs. Baldassarre et al. [8], and Pope et al. [63] proposed the idea of explainability for Graph Networks. The idea is based on the explainability methods originally designed for CNNs, such as LRP or CAM. Baldassarre et al. [8] specifically focused on the explainability of GNs using LRP. In the case of graphs with LRP, unlike images, we are interested to understand the contribution of each node or edge to the final decision. This chapter will introduce layer-wise relevance propagation in a simple neural network.

3.4.1 Conservation property of LRP

As discussed above, we want to know the contribution of each part of the input to the final prediction. Let $f : \mathbb{R}^V \rightarrow \mathbb{R}^1$ be an arbitrary classifier that maps the input of size V to a single output. The output is thresholded at zero. Therefore $f(x) > 0$ denotes the presence of the learned structure. We are interested to find out the contribution of each part of the input $x_{(d)}$ of an input x to a particular prediction $f(x)$. One possible way is to decompose the prediction $f(x)$ as a sum of terms of the separate parts of the input $x_{(d)}$:

$$f(x) \approx \sum_{d=1}^{V} R_d \tag{3.2}$$

The qualitative interpretation is that $R_d < 0$ contributes evidence against the presence of a structure which is to be classified while $R_d > 0$ contributes evidence for its presence. The purpose of layer-wise relevance propagation is to achieve a decomposition as in equation 3.2. LRP assumes that the classifier (here a neural network) can be decomposed into several layers of computation.

The first layer is the input consisting of different parts and the last layer is the real-valued prediction output of the classifier f. The l-th layer is modeled as a vector $z = \left\{z_d^{(l)}\right\}_{d=1}^{V(l)}$ with dimensionality $V(l)$. LRP assumes that we have a relevance score $R_d^{(l+1)}$ for each dimension $z_d^{(l+1)}$ of the vector z at layer $(l+1)$. The idea is to find a relevance score $R_d^{(l)}$ for each dimension $z_d^{(l)}$ of the vector z at the next layer l which is closer to the input layer such that the following equation holds.

$$f(x) = \cdots = \sum_{d\in(l+1)} R_d^{l+1} = \sum_{d\in l} R_d^{(l)} = \cdots = \sum_d R_d^{(1)} \tag{3.3}$$

Applying equation 3.3 and iterating from the output layer (classifier output $f(x)$) to the input layer x consisting of the input parts yields the desired decomposition in equation 3.2.

3.4.2 LRP in Neural Networks

Now we want to explain calculating relevance scores in a neural network. Figure 3.5 shows both forward pass for classification and backward pass for calculating relevance scores. In order to facilitate the calculation of relevance scores for each neuron, the relevance scores from higher layers are introduced as messages sent from those layers. Therefore, the relevance of a neuron i at layer l (except the last layer) is computed as follows:

$$R_i^{(l)} = \sum_{k:\, i \text{ is input for neuron } k} R_{i\leftarrow k}^{(l,l+1)} \tag{3.4}$$

The relevance of the last layer is defined as the classification score $f(x)$. Equation 3.4 checks the sum of relevance scores with respect to the output neurons from the input neuron. We can consider the other way and check the sum of relevance scores of the input neurons for the output neuron:

$$R_k^{(l+1)} = \sum_{i:\, i \text{ is input for neuron } k} R_{i\leftarrow k}^{(l,l+1)} \tag{3.5}$$

Equations 3.4 and 3.5 are the main constrains of defining Layer-wise Relevance Propagation.

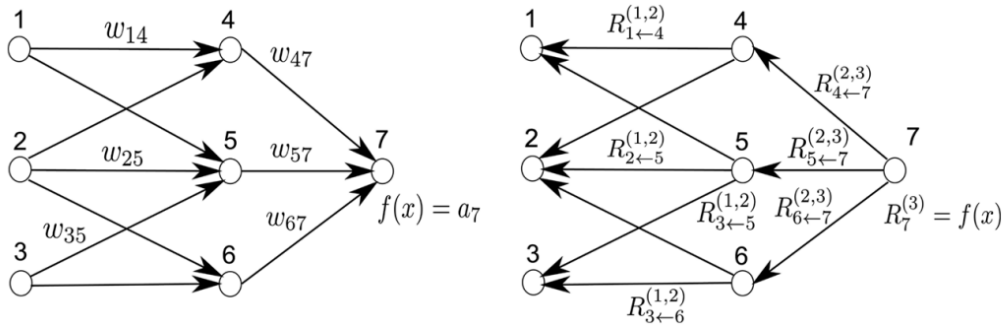


Figure 3.5: This figure is adapted from the original LRP paper [7]. In the forward pass on the left side, w_{ij} are connection weights, a_i is the activation of neuron i. In the backward pass for calculating relevance scores on the right, $R_i^{(l)}$ is the relevance of neuron i at layer l, $R_{i\leftarrow j}^{(l,l+1)}$ are relevance scores that are expressed as messages from neuron j in layer $l+1$ to neuron i in layer l. These messages are used to check whether equation 3.2 holds.

Multi-layer networks are commonly built as a set of interconnected neurons organized layer-wise. We denote neurons from layer l by x_i and neurons form layer $l+1$ by x_j. In the same manner, the summation over all neurons of layers l and $l+1$ are denoted by $\sum_i$ and $\sum_j$ respectively. A common mapping from one layer to the next one consists of a linear projection followed by a non-linear function:

$$z_{ij} = x_i w_{ij} \ , \tag{3.6}$$

$$z_j = \sum_i z_{ij} + b_j \ , \tag{3.7}$$

$$x_j = g(z_j) \tag{3.8}$$

where w_{ij} is the weight connecting neuron x_i to neuron x_j, b_j is the bias term, and g is a non-linear activation function. Common non-linear functions can be Rectified Linear Unit (ReLU) or hyperbolic tangent (tanh). One possible choice of relevance decomposition for messages from layer j to layer i is as follows:

$$R_{i\leftarrow j}^{(l,l+1)} = \frac{z_{ij}}{z_j}.R_j^{(l+1)} \tag{3.9}$$

This type of formalization guarantees the conservation properties of equation 3.3. One drawback of the equation 3.9 is that for small values z_j, $R_{i\leftarrow j}$ can take unbounded values. Two solutions provided to overcome this drawback are ε-stabilizer and $\alpha\beta$-stabilizer.

For the ε-stabilizer, let $\varepsilon \geq 0$, then the relevance scores would be as follows:

$$R_{i\leftarrow j}^{(l,l+1)} = \begin{cases} \frac{z_{ij}}{z_j+\varepsilon}.R_j^{(l+1)} & z_j \geq 0 \\ \frac{z_{ij}}{z_j-\varepsilon}.R_j^{(l+1)} & z_j < 0 \end{cases} \tag{3.10}$$

One problem with this method is that the relevance can be fully absorbed if the stabilizer ε becomes very large. For this case, we use the alternative $\alpha\beta$-stabilizer, which treats negative and positive pre-activations separately. Let $z_j^+ = \sum_i z_{ij}^+ + b_j^+$ and $z_j^- = \sum_i z_{ij}^- + b_j^-$ be negative and positive part of pre-activation respectively, where "+" and "−" denote the negative and positive values of z_{ij} and b_j. The relevance scores are then calculated as follows:

$$R_{i\leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \left(\alpha.\frac{z_{ij}^+}{z_j^+} + \beta.\frac{z_{ij}^-}{z_j^-} \right). \tag{3.11}$$

where $\alpha + \beta = 1$. This method can also control the importance of positive and negative evidence by changing the values of α and β. The complete layer-wise relevance propagation procedure for neural networks is summarized in algorithm 2.

Algorithm 2 Layer-wise relevance propagation for neural networks

1: **let** $R^{(L)} = f(x)$
2: **for** $l \in \{L-1 \ldots 1\}$ **do**
3: $\quad R_{i\leftarrow j}^{(l,l+1)}$ as in equation 3.10 or 3.11
4: $\quad R_i^{(l)} = \sum_j R_{i\leftarrow j}^{(l,l+1)}$
5: **end for**
6: **return** $\forall d : R_d^{(l)}$

3.4.3 Motivation to our problem

The LRP algorithm was initially proposed to explain image classification tasks. The output probability given by the classifier to each class is considered the relevance score in the output layer. This relevance score is then back-propagated to the input, generating a heat-map showing the most important pixels of the input image.

In our problem, we need to apply the same procedure, except that there is neither classifier nor image. The goal is to explain the policy learned by the

DRL algorithm in a robotic environment. Therefore, there are two differences between our problem and the LRP paper. First, we have a policy network instead of a classifier network. The output of the policy network is not the probability of each class. Rather it is the mean of the probability distribution from which the action is sampled. Second, our inputs are not images but graphs. Thus, the input decomposition shows the relevance of each node or edge (rather than the relevance of pixels). The following sections focus on the application of LRP to graphs in a deep reinforcement learning problem.

3.5 Deep Reinforcement Learning

In the previous sections, we discussed the graph neural networks and their necessity to our problem, how to convert a physical system to a graph, and how to use Layer-wise Relevance Propagation to provide explainability in a black box neural network. This chapter will discuss the Deep Reinforcement Learning algorithm. The algorithm used here is the state-of-the-art deep RL algorithm in robotics, Soft Actor-Critic [30], [31].

3.5.1 Reinforcement learning setting

The reinforcement learning problem can be defined as a policy search in a Markov decision process (MDP), defined by a tuple $(\mathcal{S}, \mathcal{A}, p, r)$. The state space $\mathcal{S}$ and action space $\mathcal{A}$ are assumed to be continuous, and the state transition probability $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$ represents the state transition probability of the next state $s_{t+1} \in \mathcal{S}$, given the current state $s_t \in \mathcal{S}$ and current action $a_t \in \mathcal{A}$. The environment emits a reward $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ on each transition. The state and state-action marginals of the trajectory distribution induced by a policy $\pi(a_t|s_t)$ are also denoted by $\rho_\pi(s_t)$ and $\rho_\pi(s_t, a_t)$ respectively.

3.5.2 Maximum entropy reinforcement learning

The standard objective function of reinforcement learning agent is the expected sum of rewards, and the optimal policy is defined to maximize this objective. If we denote the optimal policy as π^*, then

$$\pi^* = arg \max_\pi \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) \right] \tag{3.12}$$

The soft actor-critic algorithm's objective function is based on Maximum entropy, meaning that this algorithm, unlike traditional RL algorithms, adds an entropy term to the cumulative reward and maximizes the expectation of both cumulative reward and entropy term. This way, the policy is incentivized to explore more widely and does not converge to sub-optimal behaviours in the early stages of training. Another advantage of this entropy term is to find all

the optimal behaviours in a multi-modal environment where more than one optimal policy exists. The maximum entropy objective function would be as follows:

$$\pi^* = arg\max_{\pi} \sum_{t} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha\mathcal{H}(\pi(.|s_t))\right] \tag{3.13}$$

where α is the temperature parameter determining the relative importance of the entropy term versus the reward, which controls the stochasticity of the optimal policy. The standard RL objective can be recovered by setting α close to zero.

3.5.3 Soft Policy Iteration

Starting in the tabular case, which is a simple setting for theoretical analysis, we will discuss the policy iteration in the maximum entropy setting, which is called soft policy iteration. The policy iteration alternates between evaluating current policy and improving it.

In the policy evaluation step of soft policy iteration, the soft Q-value is computed iteratively, starting form any function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and applying Bellman backup operator $\mathcal{T}^\pi$ repeatedly as follows:

$$\mathcal{T}^\pi Q(s_t, a_t) \triangleq r(s_t, a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}\left[V(s_{t+1})\right] \tag{3.14}$$

Where the soft state value function in the above equation is calculated as follows:

$$V(s_t) = \mathbb{E}_{a_t\sim\pi}\left[Q(s_t, a_t) - \alpha\log\pi(a_t|s_t)\right] \tag{3.15}$$

In the policy improvement step, the policy is updated towards the exponential of the new soft Q-function. For this purpose, a class of policies Π is selected so that the improved policy stays in this class. For example, the class of policies can be a parameterized family of distributions like Gaussian distributions. The updated policy would be calculated using the Kullback-Leibler divergence as a projection function as follows:

$$\pi_{\text{new}} = \arg\min_{\pi'\in\Pi} D_{KL}\left(\pi'(.|s_t)\middle\|\frac{exp\left(\frac{1}{\alpha}Q^{\pi_{old}}(s_t, .)\right)}{Z^{\pi_{old}}(s_t)}\right) \tag{3.16}$$

The $Z^{\pi_{old}}(s_t)$ is just for the purpose of normalizing the distribution, and since it does not depend on the action, it does not affect the gradient. The soft policy iteration alternates between policy evaluation and improvement steps until convergence to the optimal maximum entropy policy.

3.5.4 Soft actor-critic

For large problems, we need function approximation. Therefore, soft actor-critic algorithm with neural networks as function approximator is discussed. The algorithm makes use of two soft Q-functions, parameterized by θ_1 and θ_2, to mitigate positive bias in the policy improvement step. These Q-functions are trained to optimize $J_Q(\theta_1)$ and $J_Q(\theta_2)$, where $J_Q(\theta)$ is as follows:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\Bigg[\frac{1}{2}\Bigg(Q_\theta(s_t,a_t) - \Big(r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p, a_{t+1}\sim\pi}\Big[Q_{\bar{\theta}}(s_{t+1},a_{t+1}) - \alpha\log(\pi_\phi(a_{t+1}|s_{t+1}))\Big]\Big)\Bigg)^2\Bigg] \tag{3.17}$$

In this equation, the policy is a Gaussian where the mean and the standard deviation are outputs of a neural network parameterized by ϕ, and $\mathcal{D}$ is the replay buffer to store transitions. For each pair of Q-functions with parameters θ_1 and θ_2, we have target Q-functions parameterized by $\bar{\theta_1}$ and $\bar{\theta_2}$ respectively. These two Q-functions are trained independently to optimize 3.17. The minimum of the two soft Q-functions is chosen for stochastic gradient descend in equation 3.17 and also the policy gradient update using $J_\pi(\phi)$. The policy gradient loss function $J_\pi(\phi)$ would be as follows:

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}, a_t\sim\pi_\phi}\left[\alpha\log(\pi_\phi(a_t|s_t)) - Q_\theta(s_t,a_t)\right] \tag{3.18}$$

In addition to updating the actor and critic parameters, the temperature parameter α is tuned specifically to the environment. The gradient for α is calculated using the following objective function:

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_t}\left[-\alpha\log\pi_t(a_t|s_t) - \alpha\bar{\mathcal{H}}\right] \tag{3.19}$$

The soft actor-critic is summarized in algorithm 3. It generates samples from the environment, stores them in the replay buffer, and updates neural network parameters using batches sampled from the replay buffer. The algorithm is off-policy because the policy used for generating samples and the one used for updating the parameters are not the same.

Algorithm 3 Soft Actor-Critic algorithm

Initialize networks parameterized by θ_1, θ_2, ϕ
Initialize target networks parameterized by $\bar{\theta_1} \leftarrow \theta_1$ and $\bar{\theta_2} \leftarrow \theta_2$
Initialize empty replay buffer $\mathcal{D}$
for each iteration **do**
    for each time-step t in the episode **do**
        Sample current state s_t of the environment
        $a_t \sim \pi_\phi(a_t|s_t)$ ▷ Sample action from the policy
        $s_{t+1}, r_t \sim p(s_{t+1}, r_t|s_t, a_t)$ ▷ Sample reward and next state from the environment
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$ ▷ Store the transition into the replay buffer
        if $\mathcal{D}$ has enough samples **then**
            update parameters using a batch of samples from $\mathcal{D}$ as follows:
            $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
            $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
            $\alpha \leftarrow \alpha - \lambda_\alpha \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
            $\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
        end if
    end for
end for
return θ_1, θ_2, ϕ

Chapter 4

Proposed Method

In the previous Chapter, we cover the necessary background for our methodology. We explain the structure of a graph neural network in Section 3.1 and how to convert the structure of a robot into a graph in Section 3.2. Then, we describe the Layer-wise Relevance Propagation algorithm to highlight the most relevant elements of the input to a neural network in Section 3.4. Finally, Section 3.5 explains the DRL algorithm used in this work. In this Chapter, we aim to propose our framework for interpreting the agent's policy. The general idea is to convert the observation space of a robot into a graph, use graph neural networks as function approximators in the DRL algorithm, and train the agent until convergence. After that, the LRP algorithm is applied to the learned policy network to highlight the most relevant components of a robot. This chapter covers the details of the implementation of our method.

4.1 Explainability for DRL in Robotics

In the previous chapters, we sporadically discussed the method for explaining the policy learned by a DRL agent in a robotic environment. Inspired by the explainability methods applied to graph classification tasks by Baldassarre et al. [8] and Pope et al. [63], we want to interpret the behaviour of the robot trained by a DRL algorithm. The idea is based on the similarity of the structure of a robot to graphs. In particular, the robot's observation is altered to a graph. This graph is fed to the neural network function approximators of the Soft Actor-Critic and used to output actions and update networks. After the agent is trained and the performance converged, we apply layer-wise relevance propagation to the policy network to interpret the actions. This chapter describes the details of each step.

4.1.1 Deep reinforcement learning with graph neural networks

In the reinforcement learning framework, the agent must interact with the environment, observe and get the current state s_t of the environment. It should decide what action to take based on the current state and using the policy π.

The first step toward graph operation in reinforcement learning is to convert the observation space from a vector to a graph of observations. The details of converting a robot's observation space to a graph have been argued in chapter 3.2. The observation would be in the form of:

$$s_t = G(V_t, E_t, u_t) \tag{4.1}$$

$V_t = \{\mathbf{v_t^{(i)}}\}_{i=1}^{N^v}$ is the set of node features where $v_t^{(i)}$ is the vector of node features for node i at time-step t, $E_t = \{(\mathbf{e_t^{(k)}}, r_k, s_k)\}_{k=1}^{N^e}$ is the set of edge features where $e_t^{(k)}$ is the feature vector of the edge k between the sender node s_k and receiver node r_k at time-step t, and u_t is the vector of global features of the system.

The agent applies the soft actor-critic algorithm, which comprises an actor

and a critic. Both the actor and critic use neural networks as function approximators for large state and action space environments. The second step towards applying graph neural networks to a DRL algorithm is to migrate from fully-connected neural networks (with a weak relational inductive bias) as function approximators for the policy and Q-functions to graph neural networks (with a strong relational inductive bias). For a detailed discussion on graph neural networks' operations, please refer to chapter 3.1.

4.1.2 Graph neural network architecture

According to the second step of the problem, we must describe the networks' architectures for both the policy and Q-function. For creating the graph neural network architecture, we used the `torchgraph`[1] library developed by Baldassarre et al. [8] for their work.

In this library, there are three types of graph neural network layers. These layers are according to graph operations discussed in chapter 3.1. Here we are going to explain each layer. Note that each layer receives a complete graph as input, operates on some part of the graph, and submits a graph as output. For example, the `EdgeLinear` receives a graph as input, updates its edge features and outputs a graph.

- `EdgeLinear` : This layer receives a graph as input, updates its edge features and outputs a new graph. It corresponds to the $\phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$ function as discussed in section 3.1. When we want to initialize this layer, we need to provide 5 arguments as follows (assume for each edge the triple $(\mathbf{e_t^{(k)}}, r_k, s_k)$ for $k \in 1 : N^e$):

  1. `out_features`: the length of the output graph's edge feature vector $e_t'^{(k)}$.
  2. `edge_features`: the length of the input graph's edge feature vector $e_t^{(k)}$.
  3. `sender_features`: the length of the input graph's node feature vector $v_t^{(i)}$ where $i = r_k$.

[1] https://github.com/baldassarreFe/torchgraphs

4. `receiver_features`: the length of the input graph's node feature vector $v_t^{(i)}$ where $i = s_k$.
5. `global_features`: the length of the input graph's global feature vector u_t.

- `NodeLinear` : This layer receives a graph as input, updates its node features, and outputs a new graph. It corresponds to two graph operations $\rho^{e\to v}(E_i')$ and $\phi^v(\overline{\mathbf{e}}_i', \mathbf{v}_i, \mathbf{u})$ as discussed in section 3.1. In other words, it first aggregates edge features incoming or outgoing from a specific node; it then updates the node features. The arguments are as follows (let $\{\mathbf{v_t^{(i)}}\}_{i=1}^{N^v}$ be the set of node features, and $(\mathbf{e_t^{(k)}}, r_k, s_k)$ for $k \in 1 : N^e$ be the set of edge features with their nodes specified):

1. `out_features`: the length of the output graph's node feature vector $v'^{(i)}_t$.
2. `node_features`: the length of the input graph's node feature vector $v_t^{(i)}$.
3. `incoming_features`: the length of the input graph's edge feature vector $e_t^{(k)}$ in $(\mathbf{e_t^{(k)}}, r_k, s_k)$ where $i = r_k$.
4. `outgoing_features`: the length of the input graph's edge feature vector $e_t^{(k)}$ in $(\mathbf{e_t^{(k)}}, r_k, s_k)$ where $i = s_k$.
5. `global_features`: the length of the input graph's global feature vector u_t.
6. `aggregation`: the type of $\rho^{e\to v}$ function. This aggregation can have 3 different kinds: `sum`, `avg`, and `max`.

- `GlobalLinear` : This layer receives a graph as input, updates its global features, and outputs a new graph. This layer corresponds to $\rho^{e\to u}(E')$, $\rho^{v\to u}(V')$, and $\phi^u(\overline{\mathbf{e}}', \overline{\mathbf{v}}', \mathbf{u})$ functions in the graph operations (refer to section 3.1). The arguments are as follows:

1. `out_features`: the length of the output graph's global feature vector u'_t.

2. `global_features`: the length of the input graph's global feature vector u_t.
3. `node_features`: the length of the input graph's node features.
4. `edge_features`: the length of the input graph's edge features.
5. `aggregation`: the type of aggregation functions $\rho^{e \to u}$ and $\rho^{v \to u}$. This type can be `sum`, `avg`, or `max`.

Using these graph layers, we design a graph neural network architecture commonly being used by the policy network and Q-networks of the DRL agent. This architecture is used to extract features from the observation graph. This architecture comprises two graph layers. Each layer has edge operations, node operations, and global operations. Therefore, each graph layer has one `EdgeLinear` , one `NodeLinear` , and one `GlobalLinear` layer. This architecture is summarized in figure 4.1

The architecture of the policy network and Q-Networks would be as follows:

- **Q-function**: The input to the Q-function is both a state and an action. First, the state, which is a graph, is fed to the `CommonGraph` . Then a `GlobalLinear` is applied to the `CommonGraph` 's output graph. The final graph's global features would be considered the feature vector of the input state. Then a three-layered fully-connected network would be used to combine the global feature vector output of the `GlobalLinear` layer with the action, resulting in the action-value function.

- **Gaussian Policy**: For this architecture, we fed the state graph to `CommonGraph` . Then, there are two `GlobalLinear` layers, one for the mean and another for the standard deviation of the Gaussian policy. The `CommonGraph` 's output graph is fed to mean and standard deviation `GlobalLinear` layers, producing two different graphs. The global features of the mean `GlobalLinear` 's and standard deviation `GlobalLinear` 's output graphs would be the policy distribution's mean and standard deviation, respectively.

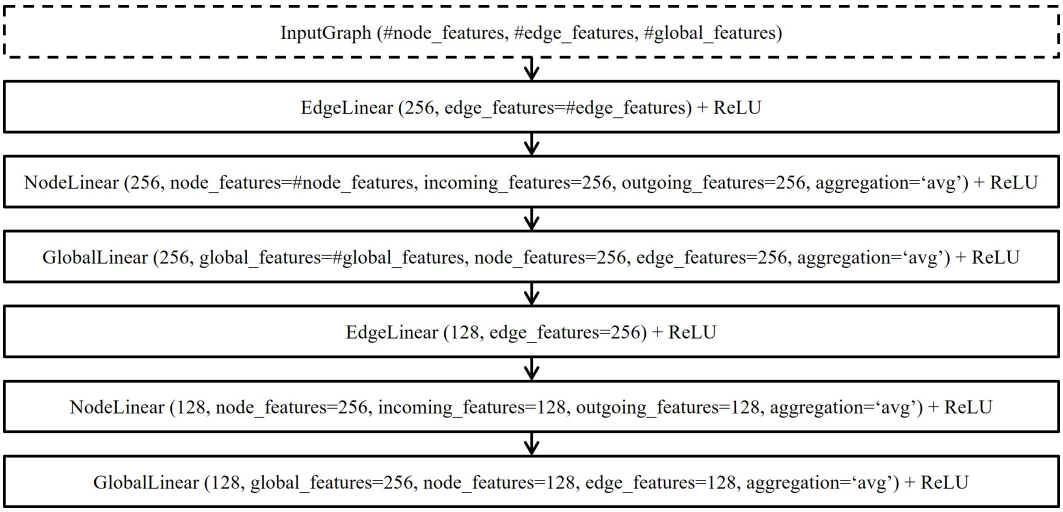


Figure 4.1: The `CommonGraph` architecture. This architecture is designed according to the algorithm 1. Only past edge features are used to update edge features during edge operations. However, for node feature update, we use edges connected to a specific node, in addition to that node's past features, to update the node's features. For global feature updates, we use both node and edge features in the update process in addition to past global features.

4.1.3 Explainability through Layer-wise Relevance Propagation

In this section, we want to explain the agent's behavior via LRP, but first, we need to break down the algorithm and analyze its parts. The agent is a soft actor-critic algorithm that comprises an actor and a critic. Therefore, it has a performance element and a learning element. A performance element converts input observation into output decisions. Thus, the actor (policy network) is the performance element of the agent. A learning element improves the performance of a performance element during the training process. The critic (Q-networks) would be the learning element of an actor-critic agent. The explanation, therefore, should be provided for the performance element, which is the learned model that performs in the environment. After the agent is trained to perform successfully in a robotic task, the LRP is applied to the policy network to project back the value of each action into the input observation, calculating relevance scores. These relevance scores can be used to analyze the impact of the robot's components on the decision-making process.

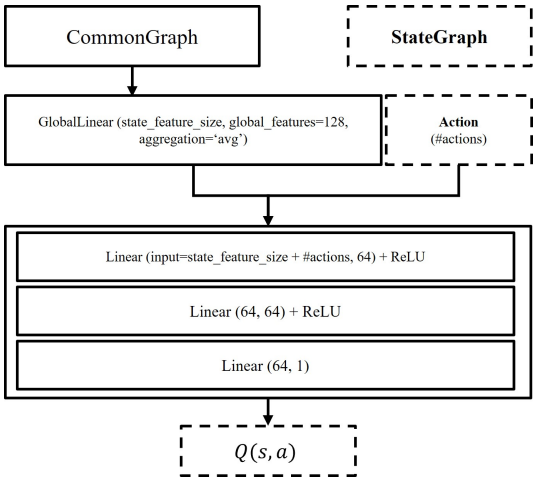


Figure 4.2: The architecture of the Q-function. The state graph is fed to the `CommonGraph` , then a `GlobalLinear` layer is applied to the output graph (note that this `GlobalLinear` layer does not have any activation function and its purpose is feature extraction). The resulting graph's global features would be used as the feature vector of the input state. This feature vector and current action are concatenated and fed into a fully connected network. The output of this fully connected network would be the Q-value.

The explanation phase starts after the training ends when the DRL algorithm has converged to the optimal policy. Unlike the training phase, where the action is sampled from a Gaussian probability distribution whose mean and standard deviation are produced by the policy net, in the evaluation phase, the actions are equal to the mean of the policy distribution. Suppose that the action at time-step t is a vector of length h,

$$a_t = [a_t^{(1)}, a_t^{(2)}, \ldots, a_t^{(h)}] \tag{4.2}$$

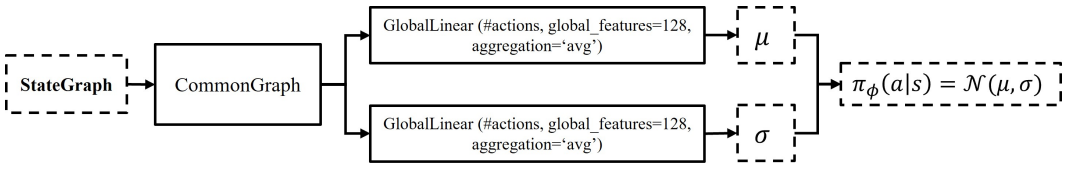


Figure 4.3: The policy network's architecture. The state is fed into the `CommonGraph` architecture. Then the output graph is fed into two separate `GlobalLinear` layers (note that these `GlobalLinear` layers do not have any activation function and their purpose is feature extraction). One of these 2 `GlobalLinear` layers is for *mean* and another is for the *standard deviation* of the policy, which is a Gaussian distribution.

In the explanation phase, at each step, the observation graph s_t is fed to the policy network π_ϕ, which outputs the action corresponding to the mean of the Gaussian distribution.

$$a_t = \pi_\phi(s_t) \tag{4.3}$$

The action is the global feature vector of the output graph of the Mean `GlobalLinear` layer in the policy net. To calculate the relevance of each action $a_t^{(i)}$ where $i \in \{0, \ldots, h-1\}$, to the input graph components, we zero out all the elements in the action vector except the element at index i, which forms the relevance score of the action i at time step t, $r_t^{(i)}$ in the output graph's global features. If $e_i \in \mathbb{R}^h$ denotes one hot vector whose elements are zero except the one at index i (which equals 1), then

$$r_t^{(i)} = a_t \cdot e_i \tag{4.4}$$

This relevance score is set to the global features of the output graph, then back-propagated to the input. If we denote the layer-wise relevance propagation operation on a neural network with $LRP()$, then

$$R_{c,t}^{(i)} = LRP(r_t^{(i)}) \quad \text{for } c \text{ in } \{0, \ldots, C-1\} \tag{4.5}$$

Where C is the number of components of the input graph, $R_{c,t}^{(i)}$ is the relevance of the action i to the component c of the input graph at time-step t given by the LRP. The LRP back-propagates the vector $r_t^{(i)}$, which is the global features of the output graph, to the input graph's components. Then, the relevance of each action to a corresponding component of the input is averaged across time steps. The evaluation phase is summarized in algorithm 4.

4.1.4 Implementation and Summary of the XRL process

The entire XRL process consists of two phases. The agent is trained on a specific environment during the first phase using the SAC algorithm with graph nets. The learning continuous until convergence to the optimal policy. At the

Algorithm 4 Calculating relevance scores for the components in the observation space.

let $R_c^{(i)} = 0$ for $c \in \{0, \ldots, C-1\}$ and $i \in \{0, \ldots, h-1\}$
let N denote the number of episodes
for each episode n in $\{0, \ldots, N-1\}$ **do**
    for each time-step t in $\{0, \ldots, T-1\}$ **do**
        Sample current state s_t of the environment
        $a_t = \pi_\phi(s_t)$ ▷ a_t equals the mean of the policy distribution
        for each element i of the action vector **do**
            $r_t^{(i)} = a_t \cdot e_i$
            $R_c^{(i)} = R_c^{(i)} + LRP(r_t^{(i)})$
        end for
    end for
end for
return $\forall c, i : R_c^{(i)}/(T \times N)$ ▷ Average of the relevance across time-steps

end of this phase, we have a learned policy network, which will be used in the explanation phase.

The second phase is the explanation phase. In this phase, the actions taken by the learned policy network are explained through layer-wise relevance propagation. The global features of the output of the Mean `GlobalLinear` layer of the policy network are used as the relevance score in the output layer. Then, this relevance score is back-propagated through graph layers to the input graph. The relevance scores from the output layer are decomposed and distributed across the input graph's nodes, edges, and global components, showing each unit's contribution to the current policy's performance.

As discussed, we used the GitHub repository developed by Baldassarre et al. [8] to build graph neural networks. In this repository, they also provided a process for calculating relevance scores. For this purpose, after training the agent using graph layers, we create the exact same network architecture with layer names shifting from `NodeLinear` to `NodeLinearRelevance` , `EdgeLinear` to `EdgeLinearRelevance` , and `GlobalLinear` to `GlobalLinearRelevance` . We call the network built using relevance layers the `RelevanceNetwork` . Then the weights of the trained policy network are loaded into the `RelevanceNetwork` for evaluation purposes and calculating relevance scores. The architecture of the

`RelevanceNetwork` is exactly the same as the original policy network, except it uses relevance layers.

Chapter 5

Experimental Analysis

In the previous chapters, we were mostly focused on introducing the idea behind our method and essential background knowledge for understanding the technique. Finally, we proposed our approach for analyzing the policy learned by a Deep Reinforcement Learning algorithm in a robotic environment. This chapter provides the results of experiments on robotic environments. Specifically, we explore the results provided by the Layer-wise Relevance Propagation on a learned policy from which we extract valuable explanations. Then in another set of experiments, we evaluate these explanations. This chapter is organized as follows: First, the setting in which we designed our experiments are explained, then the two phases of our experiments are discussed. The first phase consists of training the agent until convergence and providing a representation analysis on the observation space. Then the correctness of this analysis is evaluated in the second phase of our experiments.

5.1 Experimental Setup

The experiments are run across two simulated robotic environments in MuJoCo [79] OpenAI Gym [60]. We have two phases of experiments. In the first phase, we will train the agent until convergence using graphs as input observations and graph neural networks as function approximators for the DRL algorithm. At the end of this phase, we apply the LRP algorithm to provide a heat-map over the most important entities of the observation space with respect to each element of the action space. Using this heat-map, we will extract

the most important entities in the observation space and the most important action elements in the decision-making process. In the second phase, the importance scores calculated at the end of the first phase will be evaluated by running more experiments.

We selected `HalfCheetah-v2` and `Walker2D-v2` environments for our experiments because of the simplicity of their structure and training purposes. Nevertheless, this method can also be extended to more complex robotic settings. For the details about these environments, we refer the reader to Section 3.3. We also prepared results for the robot arm `FetchReach-v1` environment in the Appendix A. To validate the importance of each part of the robot identified by the LRP, we assume that the part is isolated from other parts. In other words, when some malfunction occurs, only one part breaks at a time. Therefore, without loss of generality, the analysis focuses only on one part at a time rather than considering a group of malfunctioning parts. Limiting the evaluation to one part at a time is sufficient because we are able to extend the analysis to more complex malfunctions, such as a group of broken parts. If the group of malfunctioning parts contains one or more important parts, in the best case, the situation would no longer be better than when only one of the important parts in that group is broken. The non-important parts would have no negative influence on the regular performance of the policy, although they can be used as a replacement for the broken part after adapting to the new dynamics.

5.2 Results and Discussion

5.2.1 Train and Explanation Phase

In this phase, we train the Soft Actor-Critic agent using Graph Networks. We run the experiments for 10 different seeds. For a detailed structure of each component of SAC, please refer to Section 4.1. After the training, we apply LRP to the learned policy to calculate the relevance scores. These scores represent the relevance of each action to each entity in the observation space at each time step. Each seed runs for 20 episodes, then the relevance

scores calculated at each time step are averaged across time steps within 20 episodes for the ten seeds. Then, these results are normalized across each action element (since the range of values for each action element at each time-step is different, hence can affect the range of relevance scores given by each action to each entity). The result is shown using a heat map indicating the relevance of each action element to each part of the observation space.

HalfCheetah-v2

Figure 5.1 represents the heat map for the `HalfCheetah-v2` environment generated by the LRP. As discussed, these scores are averaged across 20 episodes for 10 seeds. According to the `HalfCheetah-v2` documentation, the goal is for the robot to run forward as fast as possible. Therefore, the features of the *torso* entity must be critical to the policy since the goal is to increase the speed of the torso. This claim has been nicely shown by the heat-map in Figure 5.1, as the relevance score given by each action element to the torso entity is the highest across all the entities (except for the action applied to *bthigh*). Furthermore, we expect that the weight of relevance scores should be high around the diagonal since, naturally, the amount of torque applied to each joint should correspond to the state of that joint. However, in this heat map, we can only see this incident happened for the *bthigh* joint. There is also some relevance given by the *fshin* action to the *fthigh* in the observation. The reason is that since *fshin* and *fthigh* are neighbours, the state of *fthigh* can affect the amount of torque for *fshin* . This case proves one of the reasons we selected the graph structure: not only do we take into account the effect of features of each entity on the decision-making process, but we also consider their position in the structure.

From this heat map we would extract two other important information, depicted in Figures 5.3b and 5.4b. The first plot indicates the importance score given to each entity of the observation space, and the second one shows the importance of each joint in the action space provided by the LRP.

Figure 5.3b says that the most important entities to the policy are *torso* and *bthigh* , respectively. Furthermore, Figure 5.4b shows that the most critical

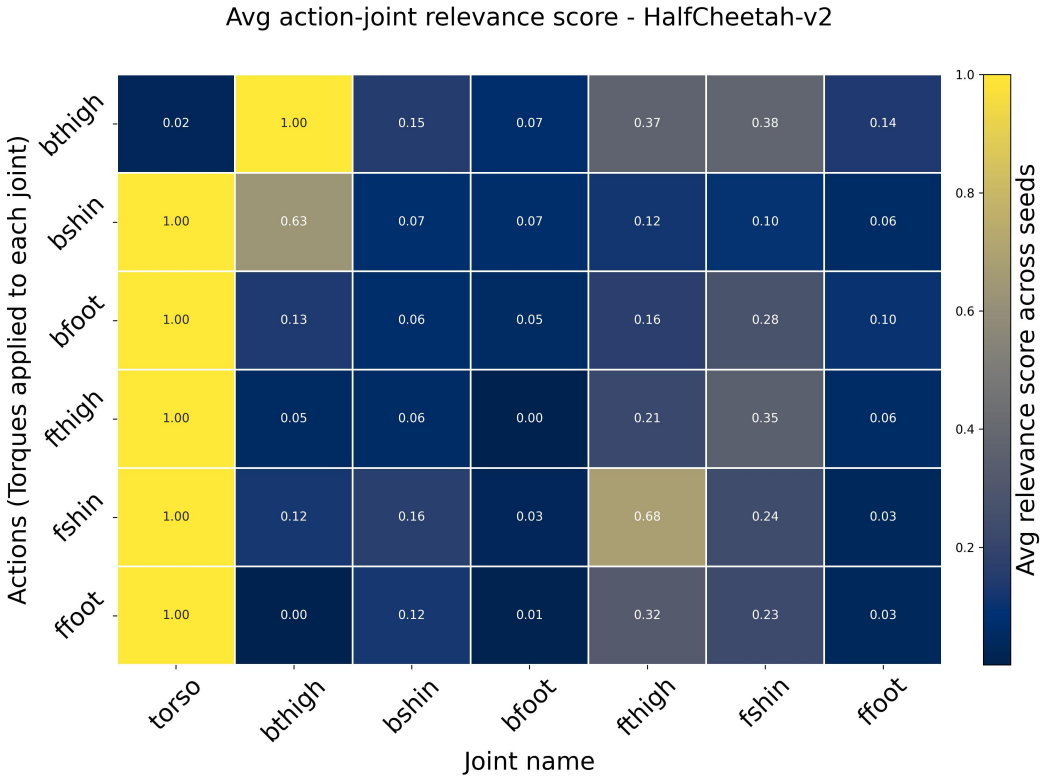


Figure 5.1: The heat-map generated by the LRP for the `HalfCheetah-v2` environment. The action elements (torques) applied to each joint are shown on the y-axis. The entities in the observation space are represented on the x-axis. The scores indicate the amount of relevance that exists between each action and each entity of the observation. These scores are averaged across 10 seeds, and each seed ran for 20 episodes. Then the scores are normalized across actions since the range of values is different for each action.

joints in the action space are *bthigh* , *bshin* , *bfoot* , and *fthigh* respectively. The correctness of these claims is evaluated in the Explanation Evaluation phase in Section 5.2.2.

Walker2D-v2

Figure 5.2 represents the heat-map produced for the `Walker2D-v2` environment by the LRP. Again these scores are calculated and averaged across 10 seeds, each running for 20 episodes. Similar to the heat map in Figure 5.1 for the `HalfCheetah-v2` , the scores given to the *torso* are the highest across all the elements in the action space. The reason is the same as `HalfCheetah-v2` – the

goal of the environment is to increase the speed of the torso. Unlike the heat-map generated for the `HalfCheetah-v2` , we can see that the relevance scores around the diagonal are the highest after the scores given to the *torso* entity for each action element. For example, for the action (torque) applied to the *thigh* joint, it depends not only on the position and velocity of the *torso* but also on the state of the *thigh* joint at each time-step. The same is true for *leg* , *thigh left* , and *leg left* . In addition to the *leg left* action, the state of the *leg left* entity in the observation space highly affects the *thigh left* action. Again, it can be explained through the vicinity of the two entities in the robot.

From this heat map, we would extract two other important information, depicted in Figures 5.5b and 5.6b. The first figure reflects the importance of each entity in the observation space, and the second shows the importance of joints in the action space.

Figure 5.5b says that the most important entity to the policy is *torso* . The importance score for the other joints is pretty small; however, the score of *thigh* and *thigh left* are higher than the remaining. Moreover, Figure 5.6b indicates that the critical joints for moving the robot faster are *foot left* and *foot* joints, respectively. The correctness of these claims is evaluated in the Explanation Evaluation phase in Section 5.2.2.

5.2.2 Explanation Evaluation Phase

In this phase, we want to evaluate the explanations provided by the LRP in each environment. Each joint plays two roles in an RL algorithm. First, its features are represented to the agent as the state of the robot at each time step. Second, an action in the form of torque is applied to that joint to move it. Therefore, there are two kinds of information extracted by the LRP:

1. **Entity Importance in the Observation Space:** Each action element gives a relevance score to each entity of the observation space. These relevance scores are averaged for each entity across actions to yield the importance of each entity to the whole decision-making process.

2. **Joint Importance in the Action Space:** The relevance scores given

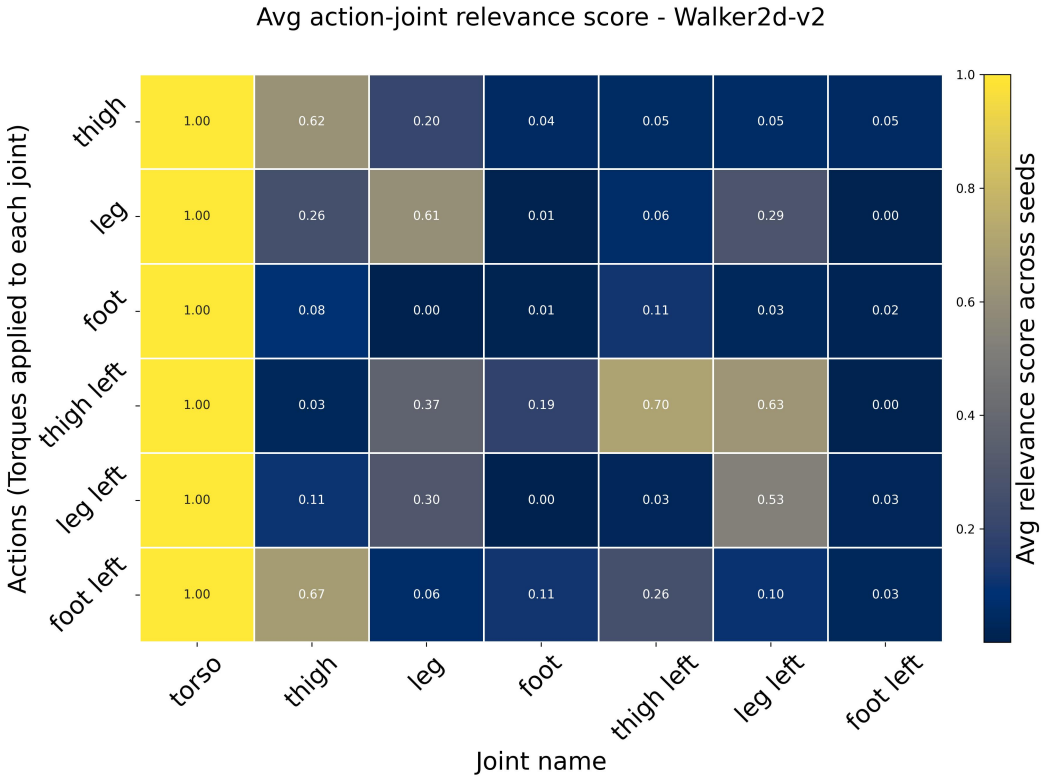


Figure 5.2: The heat-map generated by the LRP for the `Walker2D-v2` environment. The action elements (torques) applied to each joint are shown on the y-axis. The entities in the observation space are represented on the x-axis. The scores indicate the amount of relevance that exists between each action and each entity of the observation. These scores are averaged across 10 seeds, and each seed ran for 20 episodes. Then the scores are normalized across actions since the range of values is different for each action.

by each action to entities of the observation space are averaged for each action element across input entities to yield the importance of each joint in the action space.

Note that for elements of the observation space, we use the term "Entity", but for the elements of the action space, we use the term "Joint". The reason is that not all the elements of the observation space are "Joints", unlike the elements of the action space. For each environment, the evaluation targets the two kinds of information provided above.

For the importance of the entities in the observation space, we **occlude** the features of that entity in the observation space and then rerun the experiments in the new (partially observable) environment. The occlusion process corresponds to sensor failure in a joint, which causes the robot to fail to show the state of that joint at each time. Based on the amount of drop in the performance, we can evaluate the correctness of the importance scores. We expect that the amount of drop in the performance is proportional to the importance scores.

For the importance of the joints in the action space, we **block** each joint so that no torque can be applied to that joint. In other words, we remove the joint from the action space. Again, based on the amount of drop in the performance, we can measure the correctness of the importance scores for each joint in the action space. We expect that the amount of drop in the performance is proportional to the importance scores.

To ensure that the results provided by the LRP are neural network architecture independent and only depend on the policy learned by the DRL algorithm, we use fully-connected networks in this phase of experiments.

HalfCheetah-v2

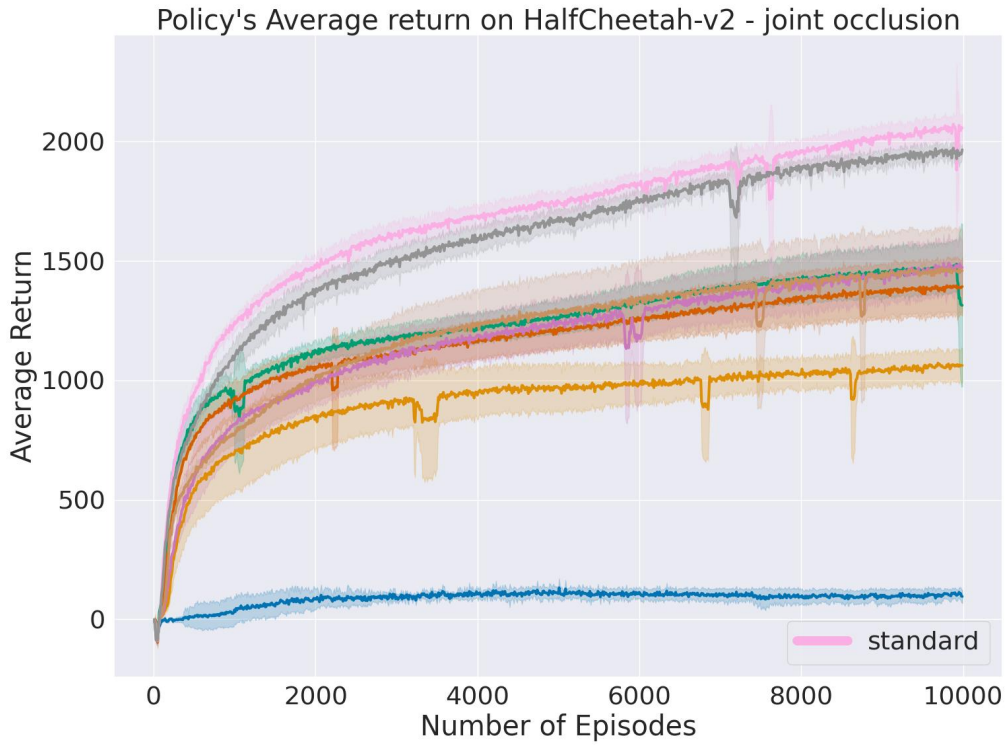
Figure 5.3 analyzes the importance scores given to each entity of the observation space in the `HalfCheetah-v2` environment. Figure 5.3b depicts the importance scores for each entity, and Figure 5.3a represents the learning curves after occluding each entity in Figure 5.3b in the observation space.

In Figure 5.3b we can see that the most important entities in the observation space are *torso* and *bthigh* . We expect that the learning curves after occluding these joints from the observation space drop significantly compared to the standard setting. This drop is evident in Figure 5.3a. Furthermore, a high drop in performance can be seen for the *fshin* during the early stages of training. However, eventually, it could recover from that and converge to the learning curves of other entities, including *bshin* , *bfoot* , and *fthigh* . For the part where the learning curves of *bshin* , *bfoot* , *fthigh* , *fshin* seem to overlap, we perform a statistical t-test with a confidence interval of 95%. Figure 5.3c reflects the P-values for the significance test between each pair of learning curves. For $P < 0.05$ the learning curves are significantly different. It can be deducted from this t-test that the learning curves of *fshin* and *bshin* , and *fshin* and *fthigh* are not significantly different. However, for *fshin* the drop in performance during the early stages of training is higher than in the other two. Although their learning curves are significantly different for *bshin* and *fthigh* , the P-value is close to 0.05, so their performance is close to each other (as it can be deducted from their importance scores). For *ffoot* we can say that the importance score is unexpectedly higher than *bshin* , *bfoot* , and *fthigh* .

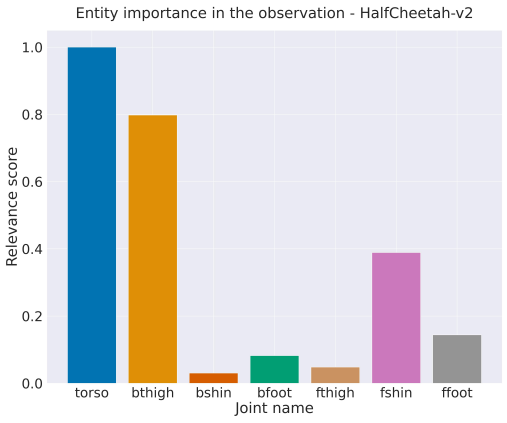
Figure 5.4 analyzes the importance of each joint in the action space according to the scores provided by the LRP. Figure 5.4b says that the most important joints are *bthigh* , *bshin* , *bfoot* , and *fthigh* respectively. These results are evaluated in Figure 5.4a in which each joint in Figure 5.4b is blocked, and the performance in the new setting is analyzed. It is clear that the drop in performance in *bthigh* is proportional to its importance score. The same is true for *bshin* , *bfoot* , and *fthigh* . However, we expected that the importance score of *bfoot* would be higher than the other two, but it is negligible since the learning curves of these three joints are pretty close to each other, as their importance scores are. The three joints *fthigh* , *fshin* , and *ffoot* having the least importance scores have learning curves that are not significantly different. This claim can be deduced from the statistical t-test analysis in Figure 5.4c.

It should be noted that the learning curves in Figures 5.3a and 5.4a are

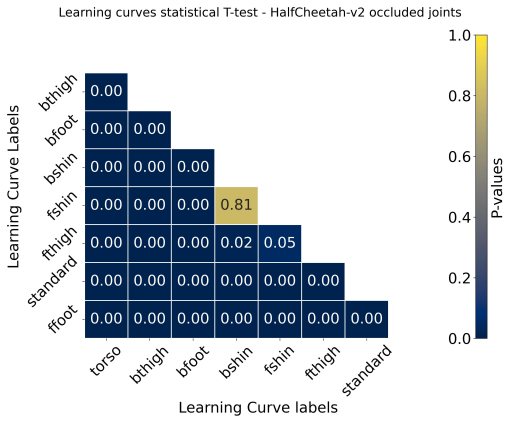
averaged across 10 seeds, and the shaded parts reflect the standard error of each curve.



(a)

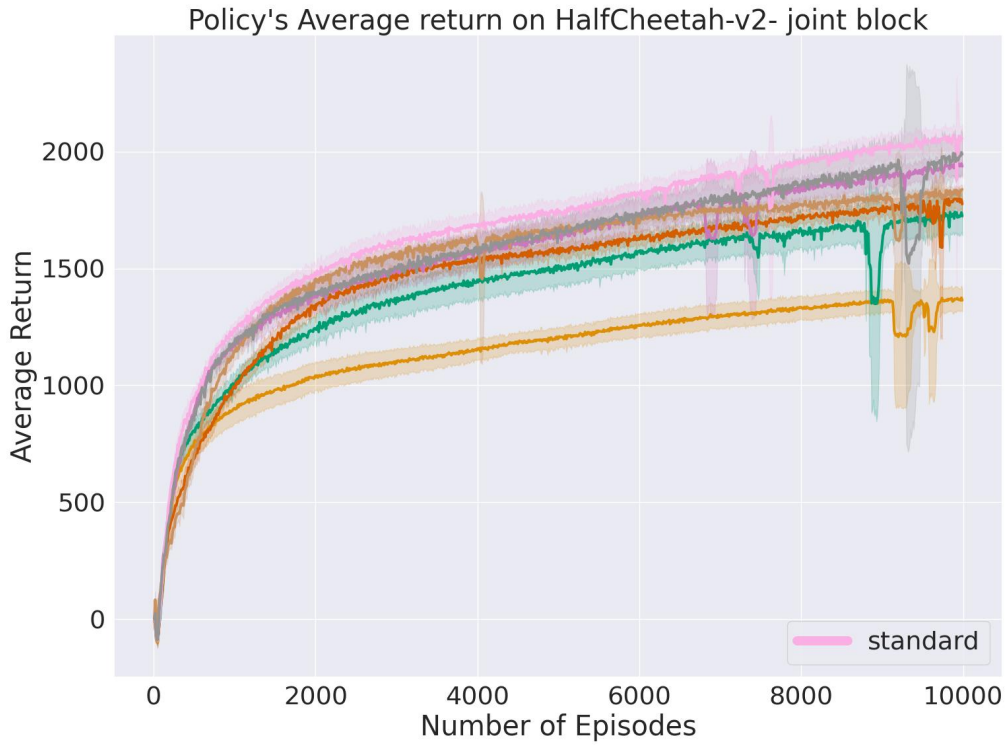


(b)

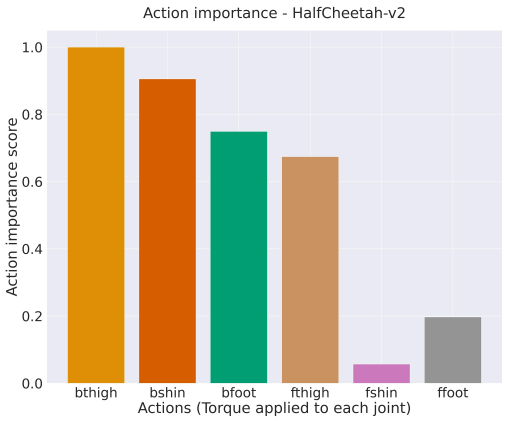


(c)

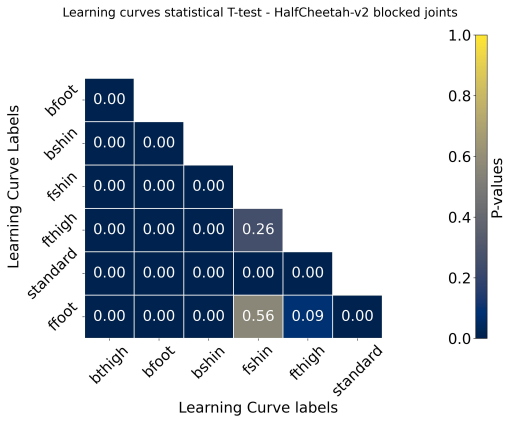
Figure 5.3: Evaluating the importance scores given by LRP to each entity of the `HalfCheetah-v2` robot. a) shows learning curves of training on the new environment after occluding features of each entity in the observation space. The color of learning curves corresponds to the color of bar plots in (b). b) reflects the entity importance score in the observation space calculated by LRP. c) indicates whether each pair of curves in (a) are significantly different or not via statistical t-test. Learning curves are significantly different For $P < 0.05$)



(a)



(b)



(c)

Figure 5.4: Evaluating the importance scores given by LRP to each element of the action space in `HalfCheetah-v2` robot. a) shows learning curves of training on the new environment after blocking each joint. The color of learning curves corresponds to the color of bar plots in (b). b) reflects the importance score of the joints in the action space calculated by LRP. c) indicates whether each pair of curves in (a) are significantly different or not via statistical t-test. Learning curves are significantly different For $P < 0.05$)

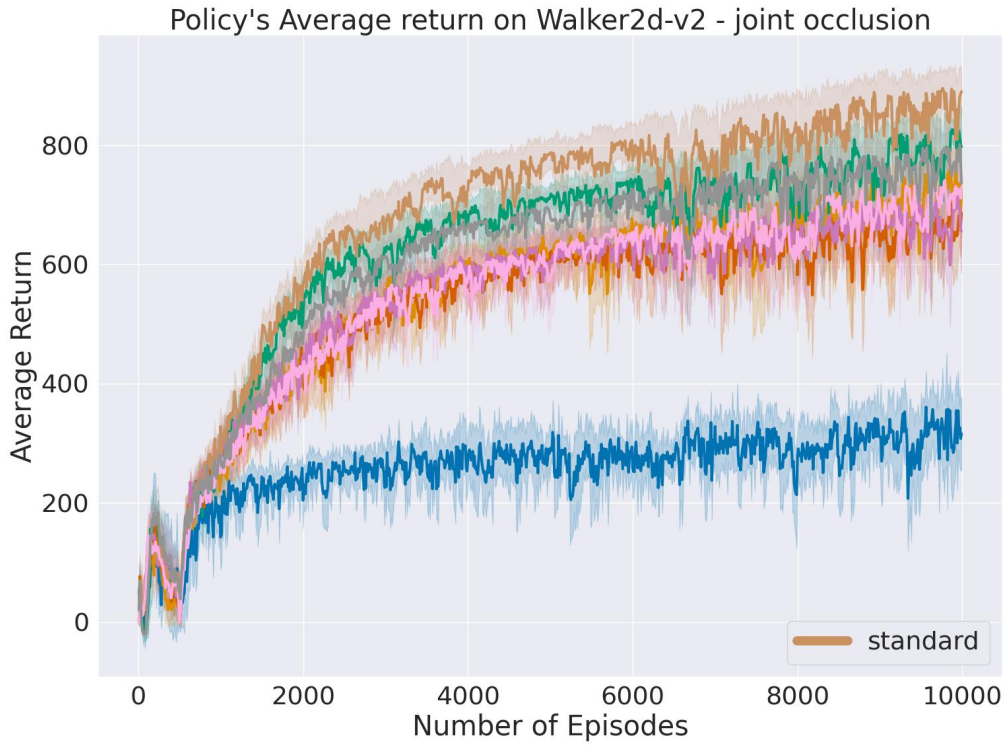
Walker2D-v2

Figure 5.5 reflects the results of experiments evaluating the entity's importance in the observation space for the `Walker2D-v2` environment. Figure 5.5b says that the most important entity in the observation space is the *torso* . The scores for entities other than *torso* are pretty low and close to each other, with *thigh* and *thigh left* having equally the highest score among them. As expected, by occluding the *torso* from the observation space, the agent could not learn a policy at all, therefore had the most significant drop in performance in Figure 5.5a. The learning curves overlap for the *thigh* and *thigh left* , which can be shown by Figure 5.5c. Figure 5.5c reflects the result of statistical t-test with 95% confidence interval between each pair of learning curves in Figure 5.5a. The highest importance score after *thigh* and *thigh left* belongs to the *leg* joint. The learning curves of the *thigh* , *thigh left* , and *leg* are not significantly different, as shown in Figure 5.5c. The *foot* and *foot left* have the least drop in performance compared to the standard setting, which can be deduced from their importance score as well. Nevertheless, the importance score for the *leg left* is unexpected because, based on its learning curve, we expect its score to be noticeable, similar to *thigh* , *thigh left* , and *leg* .

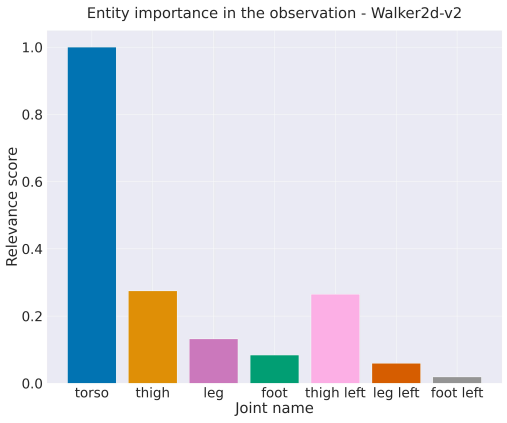
Figure 5.6 shows the results of evaluating importance scores for each joint in the action space for the `Walker2D-v2` environment. As indicated by Figure 5.5b, *foot left* and *foot* joints are the most important ones. Figure 5.6a proves the correctness of this claim by showing that the amount of drop in performance after blocking *foot left* and *foot* joints is proportional to their importance scores. For *leg* and *leg left* the amount of drop in performance is close to each other, as their importance scores are. The unusual thing here is that the importance score for *thigh left* is high, but after blocking this joint, the performance became even better than the standard setting. This irregularity says that LRP fails to give a correct importance score to joints having unexpected behavior, or redundant to learning a good policy. The same conclusion can be applied to the *thigh* joint.

It should be noted that the learning curves in Figures 5.5a and 5.6a are

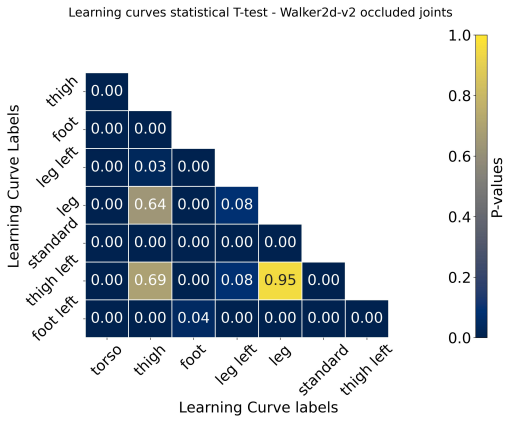
averaged across 30 seeds, and the shaded parts reflect the standard error of each curve.



(a)

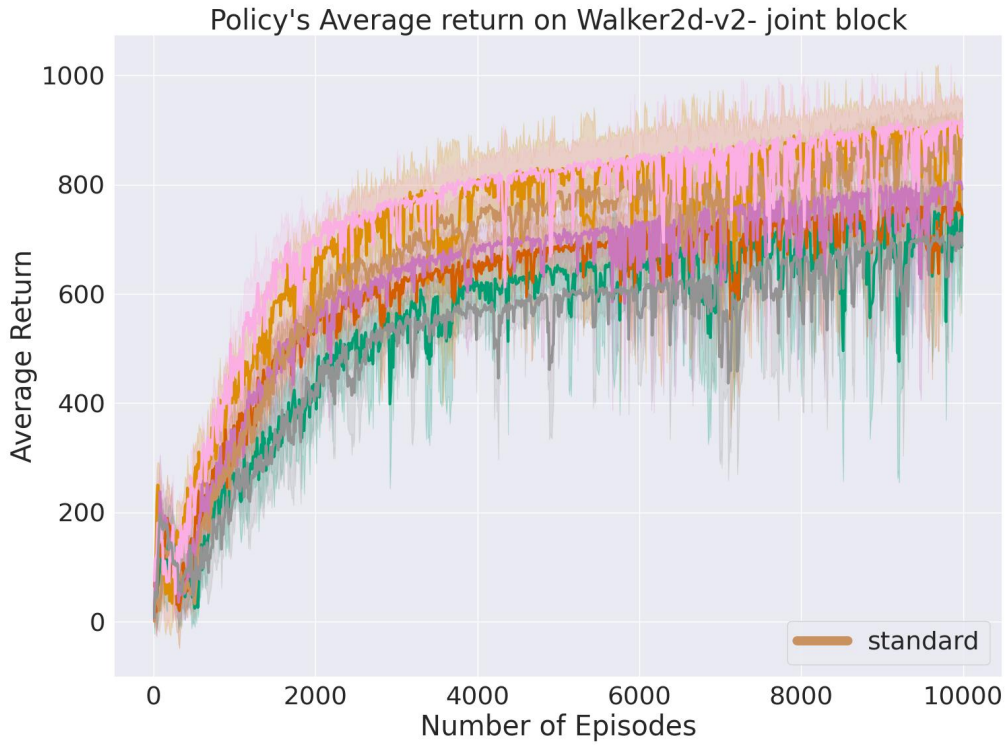


(b)

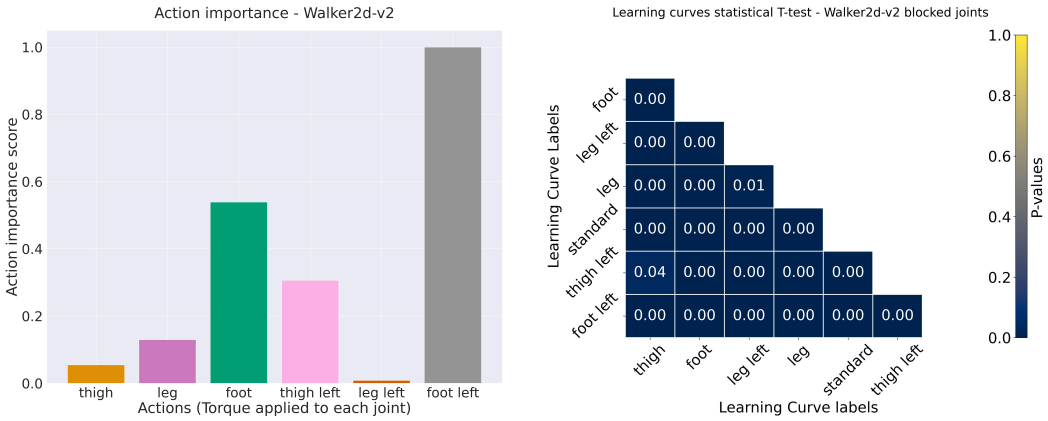


(c)

Figure 5.5: Evaluating the importance scores given by LRP to each entity of the `Walker2D-v2` robot. a) shows learning curves of training on the new environment after occluding features of each entity in the observation space. The color of learning curves corresponds to the color of bar plots in (b). b) reflects the entity importance score in the observation space calculated by LRP. c) indicates whether each pair of curves in (a) are significantly different or not via statistical t-test. Learning curves are significantly different For $P < 0.05$)



(a)



(b) (c)

Figure 5.6: Evaluating the importance scores given by LRP to each element of the action space in `Walker2D-v2` robot. a) shows learning curves of training on the new environment after blocking each joint. The color of learning curves corresponds to the color of bar plots in (b). b) reflects the importance score of the joints in the action space calculated by LRP. c) indicates whether each pair of curves in (a) are significantly different or not via statistical t-test. Learning curves are significantly different For $P < 0.05$)

Chapter 6

Conclusion and Future work

6.1 Conclusion

We propose a novel technique for interpreting the deep reinforcement learning black box modules in robotic environments. The idea is to find the most contributing elements of the observation space to the decision-making process. To do so, we first decompose the robot into entities with a relationship between each pair. Therefore, we selected graphs to represent the observation to take advantage of the strong relational inductive bias of graph neural network architectures and consider each entity's position relative to other entities for the interpretation phase. To find the most contributing components of the robot to the decision-making process, we apply the Layer-wise Relevance Propagation algorithm.

We aim to evaluate our approach on two well-known MuJoCo robotic environments, namely `HalfCheetah-v2` and `Walker2D-v2` . Our empirical results prove that our approach can successfully find the most important entities of the robot's observation space in the decision-making process. Moreover, this method could successfully identify the most critical joints for the action space to reach the target of the environment.

The results provided by our technique can be used for debugging purposes. Using this method, one can analyze which components in the robot are being used more often and what entities in the robot play an important role in taking action. Another application is in machine maintenance, where our method, after diagnosing which part of the robot malfunctions, can tell whether the

RL agent can recover from that fault or the malfunctioning part needs to be replaced. In other words, it tells us about the severity of the damage. Similarly, suppose one chooses the machine to adapt to the new dynamics. In that case, it can give us some intuition about the adaptation process by comparing the relevance scores before and after adaptation. Furthermore, this method can answer questions about how the behavior of two policies is different and the reason that one policy performs better than the other.

6.2 Future directions

One future approach is to use the LRP scores as information for data-driven machine maintenance or informed adaptation to the new dynamics. When a robot malfunctions and we diagnose the broken part, using the importance of that part in both the observation and the action spaces, the RL agent can find a better way to recover from that fault. In addition, one can explain the adaptation process in a group of tasks dealing with robot maintenance.

Another direction is to visualize and analyze the LRP scores during training. This can give us intuition about how the RL agent finds the optimal behavior by analyzing the change in the importance of each entity every number of time steps.

One important future work can focus on informed transfer learning. Knowing how important a component is to the RL agent, we can decide whether the policy learned in a normal environment can be applied in another environment with different dynamics.

References

[1] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," *IEEE access*, vol. 6, pp. 52 138–52 160, 2018.

[2] D. Amir and O. Amir, "Highlights: Summarizing agent behavior to people," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 1168–1176.

[3] M. Andrychowicz, F. Wolski, A. Ray, *et al.*, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.

[4] O. M. Andrychowicz, B. Baker, M. Chociej, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[5] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010.

[6] A. Atrey, K. Clary, and D. Jensen, "Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning," *arXiv preprint arXiv:1912.05743*, 2019.

[7] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, e0130140, 2015.

[8] F. Baldassarre and H. Azizpour, "Explainability techniques for graph convolutional networks," *arXiv preprint arXiv:1905.13686*, 2019.

[9] P. W. Battaglia, J. B. Hamrick, V. Bapst, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[10] R. A. Berk and J. Bleich, "Statistical procedures for forecasting criminal behavior: A comparative assessment," *Criminology & Pub. Pol'y*, vol. 12, p. 513, 2013.

[11] B. Beyret, A. Shafti, and A. A. Faisal, "Dot-to-dot: Explainable hierarchical reinforcement learning for robotic manipulation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 5014–5019.

[12] M. Bojarski, D. Del Testa, D. Dworakowski, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[13] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 1721–1730.

[14] Z. Che, S. Purushotham, R. Khemani, and Y. Liu, "Interpretable deep models for icu outcome prediction," in *AMIA annual symposium proceedings*, American Medical Informatics Association, vol. 2016, 2016, p. 371.

[15] J. Chen, S. E. Li, and M. Tomizuka, "Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[16] G. Cideron, M. Seurin, F. Strub, and O. Pietquin, "Self-educated language agent with hindsight experience replay for instruction following," 2019.

[17] F. Cruz, R. Dazeley, and P. Vamplew, "Memory-based explainable reinforcement learning," in *Australasian Joint Conference on Artificial Intelligence*, Springer, 2019, pp. 66–77.

[18] F. Cruz, R. Dazeley, P. Vamplew, and I. Moreira, "Explainable robotic systems: Understanding goal-driven actions in a reinforcement learning scenario," *Neural Computing and Applications*, pp. 1–18, 2021.

[19] G. Dao, I. Mishra, and M. Lee, "Deep reinforcement learning monitor for snapshot recording," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2018, pp. 591–598.

[20] S. Doncieux, N. Bredeche, L. L. Goff, *et al.*, "Dream architecture: A developmental approach to open-ended learning in robotics," *arXiv preprint arXiv:2005.06223*, 2020.

[21] S. Doncieux, D. Filliat, N. Dıéaz-Rodrıéguez, *et al.*, "Open-ended learning: A conceptual framework based on representational redescription," *Frontiers in neurorobotics*, vol. 12, p. 59, 2018.

[22] U. Ehsan, B. Harrison, L. Chan, and M. O. Riedl, "Rationalization: A neural machine translation approach to generating natural language explanations," in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, 2018, pp. 81–87.

[23] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[24] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*, Springer, 1982, pp. 267–285.

[25] M. Garnelo, K. Arulkumaran, and M. Shanahan, "Towards deep symbolic reinforcement learning," *arXiv preprint arXiv:1609.05518*, 2016.

[26] M. Gevrey, I. Dimopoulos, and S. Lek, "Review and comparison of methods to study the contribution of variables in artificial neural network models," *Ecological modelling*, vol. 160, no. 3, pp. 249–264, 2003.

[27] O. Gottesman, J. Futoma, Y. Liu, *et al.*, "Interpretable off-policy evaluation in reinforcement learning by highlighting influential transitions," in *International Conference on Machine Learning*, PMLR, 2020, pp. 3658–3667.

[28] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," in *International Conference on Machine Learning*, PMLR, 2018, pp. 1792–1801.

[29] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, "Xai—explainable artificial intelligence," *Science Robotics*, vol. 4, no. 37, 2019.

[30] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[31] T. Haarnoja, A. Zhou, K. Hartikainen, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[32] J. Haspiel, N. Du, J. Meyerson, *et al.*, "Explanations and expectations: Trust building in automated vehicles," in *Companion of the 2018 ACM/IEEE international conference on human-robot interaction*, 2018, pp. 119–120.

[33] B. Hayes and J. A. Shah, "Improving robot controller transparency through autonomous policy explanation," in *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, IEEE, 2017, pp. 303–312.

[34] D. Hein, S. Udluft, and T. A. Runkler, "Interpretable policies for reinforcement learning by genetic programming," *Engineering Applications of Artificial Intelligence*, vol. 76, pp. 158–169, 2018.

[35] A. Henelius, K. Puolamäki, and A. Ukkonen, "Interpreting classifiers through attribute interactions in datasets," *arXiv preprint arXiv:1707.07576*, 2017.

[36] A. Heuillet, F. Couthouis, and N. Dıéaz-Rodrıéguez, "Explainability in deep reinforcement learning," *Knowledge-Based Systems*, vol. 214, p. 106 685, 2021.

[37] A. Holzinger, C. Biemann, C. S. Pattichis, and D. B. Kell, "What do we need to build explainable ai systems for the medical domain?" *arXiv preprint arXiv:1712.09923*, 2017.

[38] C. Howell, "A framework for addressing fairness in consequential machine learning," in *Proc. FAT Conf., Tuts.*, 2018, pp. 1–2.

[39] S. H. Huang, K. Bhatia, P. Abbeel, and A. D. Dragan, "Establishing appropriate trust via critical states," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 3929–3936.

[40] T. Huber, D. Schiller, and E. André, "Enhancing explainability of deep reinforcement learning through selective layer-wise relevance propagation," in *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, Springer, 2019, pp. 188–202.

[41] R. Iyer, Y. Li, H. Li, M. Lewis, R. Sundar, and K. Sycara, "Transparency and explanation in deep reinforcement learning neural networks," in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, 2018, pp. 144–150.

[42] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez, "Explainable reinforcement learning via reward decomposition," in *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*, 2019.

[43] D. Kalashnikov, A. Irpan, P. Pastor, *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on Robot Learning*, PMLR, 2018, pp. 651–673.

[44] G. J. Katuwal and R. Chen, "Machine learning model interpretability for precision medicine," *arXiv preprint arXiv:1610.09045*, 2016.

[45] W. Knight, "The us military wants its autonomous machines to explain themselves," *Retrieved February*, vol. 23, p. 2018, 2017.

[46] I. Lage, D. Lifschitz, F. Doshi-Velez, and O. Amir, "Exploring computational user models for agent policy summarization," in *IJCAI: proceedings of the conference*, NIH Public Access, vol. 28, 2019, p. 1401.

[47] M. Landajuela, B. K. Petersen, S. Kim, *et al.*, "Discovering symbolic policies with deep reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2021, pp. 5979–5989.

[48] Y. LeCun, B. Boser, J. S. Denker, *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[49] T. Lesort, N. Dıéaz-Rodrıéguez, J.-F. Goudou, and D. Filliat, "State representation learning for control: An overview," *Neural Networks*, vol. 108, pp. 379–392, 2018.

[50] T. Lesort, M. Seurin, X. Li, N. Dıéaz-Rodrıéguez, and D. Filliat, "Deep unsupervised state representation learning with robotic priors: A robustness analysis," in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–8.

[51] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[52] D. T. Limited, *Modeling*, https://mujoco.readthedocs.io/en/latest/modeling.html, Accessed: 2022-01-29, 2022.

[53] Z. Lin, K.-H. Lam, and A. Fern, "Contrastive explanations for reinforcement learning via embedded self predictions," *arXiv preprint arXiv:2010.05180*, 2020.

[54] G. Liu, O. Schulte, W. Zhu, and Q. Li, "Toward interpretable deep reinforcement learning with linear model u-trees," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2018, pp. 414–429.

[55] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere, "Explainable reinforcement learning through a causal lens," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 2493–2500.

[56] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, 2016, pp. 50–56.

[57] T. M. Mitchell, *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research ..., 1980.

[58] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[59] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[60] OpenAI, *Getting started with gym*, https://gym.openai.com/docs/, Accessed: 2022-01-29, 2021.

[61] ——, *Half cheetah*, https://www.gymlibrary.ml/environments/mujoco/half_cheetah/, Accessed: 2022-08-16, 2022.

[62] ——, *Walker2d*, https://www.gymlibrary.ml/environments/mujoco/walker2d/, Accessed: 2022-08-16, 2022.

[63] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann, "Explainability methods for graph convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 772–10 781.

[64] E. Puiutta and E. M. Veith, "Explainable reinforcement learning: A survey," in *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, Springer, 2020, pp. 77–95.

[65] A. Raffin, A. Hill, R. Traoré, T. Lesort, N. Dıéaz-Rodrıéguez, and D. Filliat, "S-rl toolbox: Environments, datasets and evaluation metrics for state representation learning," *arXiv preprint arXiv:1809.09369*, 2018.

[66] ——, "Decoupling feature extraction from policy learning: Assessing benefits of state representation learning in goal based robotics," *arXiv preprint arXiv:1901.08651*, 2019.

[67] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, *et al.*, "Graph networks as learnable physics engines for inference and control," in *International Conference on Machine Learning*, PMLR, 2018, pp. 4470–4479.

[68] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

[69] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[70] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[71] P. Sequeira and M. Gervasio, "Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations," *Artificial Intelligence*, vol. 288, p. 103 367, 2020.

[72] P. Sequeira, E. Yeh, and M. T. Gervasio, "Interestingness elements for explainable reinforcement learning through introspection.," in *IUI workshops*, vol. 1, 2019.

[73] A. Silva, T. Killian, I. D. J. Rodriguez, S.-H. Son, and M. Gombolay, "Optimization methods for interpretable differentiable decision trees in reinforcement learning," *arXiv preprint arXiv:1903.09338*, 2019.

[74] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[75] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[76] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[77] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[78] S. Tan, R. Caruana, G. Hooker, and Y. Lou, "Detecting bias in black-box models using transparent model distillation," *arXiv preprint arXiv:1710.06169*, 2017.

[79] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 5026–5033.

[80] N. Topin, S. Milani, F. Fang, and M. Veloso, "Iterative bounding mdps: Learning interpretable policies via non-interpretable methods," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 9923–9931.

[81] R. Traoré, H. Caselles-Dupré, T. Lesort, *et al.*, "Discorl: Continual reinforcement learning via policy distillation," *arXiv preprint arXiv:1907.05855*, 2019.

[82] S. Volodin, "Causeoccam: Learning interpretable abstract representations in reinforcement learning environments via model sparsity," Tech. Rep., 2021.

[83] J. Wang, Y. Zhang, T.-K. Kim, and Y. Gu, "Shapley q-value: A local reward approach to solve global reward games," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 7285–7292.

[84] T. Wang, R. Liao, J. Ba, and S. Fidler, "Nervenet: Learning structured policy with graph neural networks," in *International Conference on Learning Representations*, 2018.

[85] X. Wang, S. Yuan, H. Zhang, M. Lewis, and K. Sycara, "Verbal explanations for deep reinforcement learning neural networks with attention on extracted features," in *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, IEEE, 2019, pp. 1–7.

[86] L. Weitkamp, E. van der Pol, and Z. Akata, "Visual rationalizations in deep reinforcement learning for atari games," in *Benelux Conference on Artificial Intelligence*, Springer, 2018, pp. 151–165.

[87] H. Yau, C. Russell, and S. Hadfield, "What did you think would happen? explaining agent behaviour through intended outcomes," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 375–18 386, 2020.

[88] T. Zahavy, N. Ben-Zrihem, and S. Mannor, "Graying the black box: Understanding dqns," in *International Conference on Machine Learning*, PMLR, 2016, pp. 1899–1908.

[89] V. Zambaldi, D. Raposo, A. Santoro, *et al.*, "Relational deep reinforcement learning," *arXiv preprint arXiv:1806.01830*, 2018.

[90] J. Zhang, S. A. Bargal, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff, "Top-down neural attention by excitation backprop," *International Journal of Computer Vision*, vol. 126, no. 10, pp. 1084–1102, 2018.

[91] L. Zhang, X. Li, M. Wang, and A. Tian, "Off-policy differentiable logic reinforcement learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2021, pp. 617–632.

[92] Q. Zhang, Y. N. Wu, and S.-C. Zhu, "Interpretable convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8827–8836.

[93] G. Zheng, F. Zhang, Z. Zheng, *et al.*, "Drn: A deep reinforcement learning framework for news recommendation," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 167–176.

[94] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921–2929.

[95] Z. Zhou, X. Li, and R. N. Zare, "Optimizing chemical reactions with deep reinforcement learning," *ACS central science*, vol. 3, no. 12, pp. 1337–1344, 2017.

Appendix A

FetchReach-v1 Results

A.1 Train and Explanation Phase

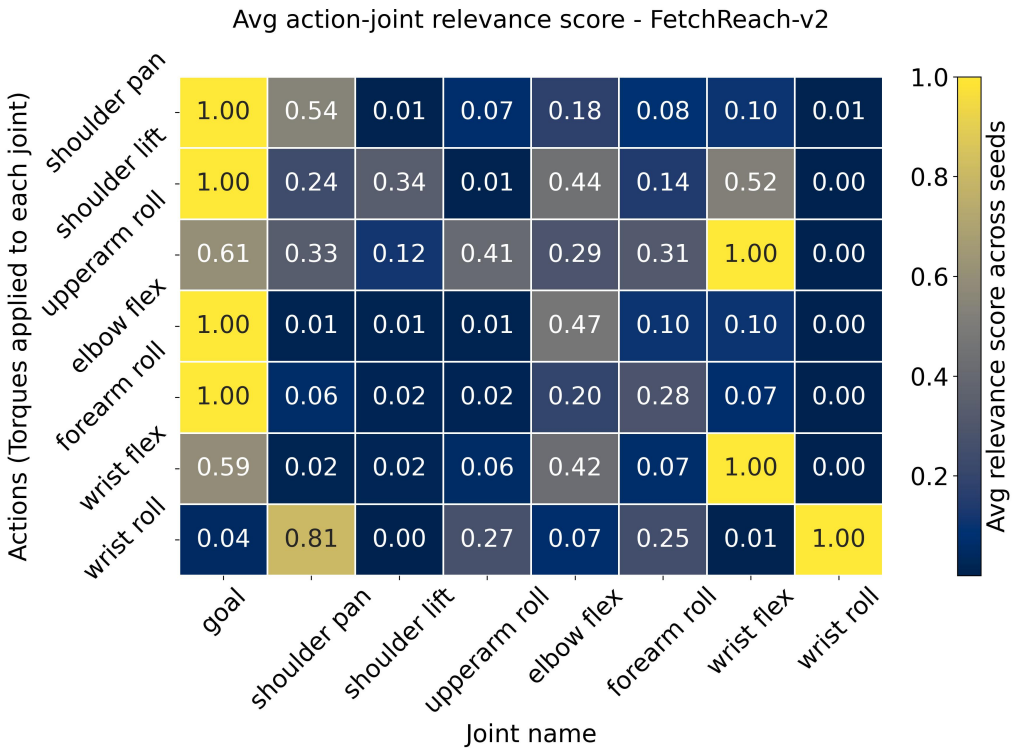


Figure A.1: The LRP heat map for action-entity relevance score. The y-axis and x-axis show elements of the action and entities of the observation, respectively. Since joints have different characteristics and possible amounts of torque, the actions have different ranges. Therefore, we normalize the relevance scores for each action across all the observation entities by dividing by the maximum score given by that action.

Figure A.1 shows the heat map generated by the LRP for `FetchReach-v1` . The position of the `goal` receives a high score across all (except wrist_roll) actions. This is because all the torques should be adjusted in a way to position the end-effector at the `goal` . Ignoring the `goal` column, similar to `Walker2D-v2` 's heat map, we can see a diagonal pattern. The highest scores for shoulder_pan , elbow_flex , forearm_roll , wrist_flex , and wrist_roll actions belong to their corresponding joint entities in the observation space as expected. Although for shoulder_lift and upperarm_roll actions the scores on the diagonal are unexpectedly not the highest, still the scores given to their corresponding joint entities in the observation space are relatively high. This unexpected result might be explained using the graph structure and the vicinity of joints: for shoulder_lift and upperarm_roll actions, the relevance scores are distributed across multiple entities in the observation space.

According to the *entity importance* plot (top-left bar plot in Figure A.2), the most important entity to the policy is `goal` . Other than `goal` , shoulder_pan , forearm_roll , wrist_roll , and upperarm_roll have relatively high importance respectively. Other entities have pretty small importance scores. Moreover, the *action importance* plot (bottom-left bar plot in Figure A.2) indicates that the critical joints for positioning the robot's end-effector to the `goal` are elbow_flex , wrist_flex , and forearm_roll joints, respectively.

A.2 Explanation Evaluation Phase

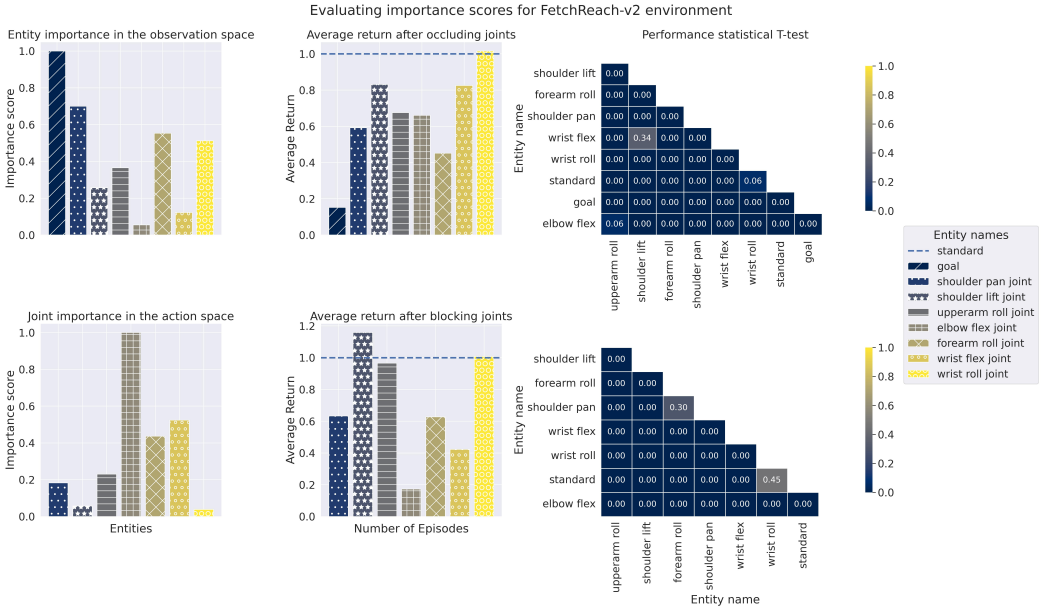


Figure A.2: Evaluating explanation for the `FetchReach-v1` . Upper-left: entity importance in the observation, upper-middle: final behavior performance after occluding each entity, upper-right: significancy test for the final behavior after occlusion, lower-left: joint importance in the action, lower-middle: final behavior performance after blocking each joint, lower-right: significancy test for the final behaviors after blocking.

First, we focus on analyzing the observation entity importance in the upper-row plots of Figure A.2. The upper-left bar plot indicates that the most important entity is the `goal` . As expected, when occluding the `goal` , the performance drops significantly, as indicated in the upper-middle plot. After `goal` , shoulder_pan and forearm_roll entities have the highest scores. Their corresponding performance bars show a proportional amount of drop in performance after occlusion. We expect the wrist_roll joint to receive a relatively low importance score based on its performance bar. Although the wrist_roll joint seemed to be critical according to its importance score, if we look at the heat map of the `FetchReach-v1` in Figure A.1, the action applied to this joint gave a relatively low score to the `goal` . Thus, we can conclude that this joint does not contribute to reaching the goal. That is why after occluding it, the

performance did not change. The high score of wrist_roll entity is only because of the score given by the wrist_roll action. The importance score of the upperarm_roll entity can also be verified by its performance bar. However, since the upperarm_roll and elbow_flex performance bars are approximately equal, we expect a high score for the elbow_flex , unlike its current importance score. It remains shoulder_lift and wrist_flex entities that, as clear from their performance bars, the amount of drop in their performance can imply their importance score.

The joint importance in the action space evaluation is reflected in the lower-row plots of Figure A.2. As indicated on the lower-left bar plot, the most critical joints to the actions are elbow_flex , wrist_flex , and forearm_roll joints, respectively, and are highly strategic for reaching the goal. The noticeable drop in the performance after blocking these three joints, shown in their performance bar in the lower-middle plot, implies their importance. As discussed, the wrist_roll joint does not contribute to reaching the goal. Therefore neither its occlusion nor its blockage affects the performance. For upperarm_roll and shoulder_lift joints, the drop in their performance bar compared to the standard setting can be correctly explained by their importance score. Nevertheless, the LRP fails to explain the performance improvement after the shoulder_lift joint's blockage. For shoulder_pan joint, we expect that LRP gives an importance score approximately equal to the forearm_roll joint because their performance is nearly the same.

Appendix B

Mujoco physics engine

This section contains the XML models representing the kinematic tree of the robots in the Ant-v2 and FetchReach-v1 environments.

B.1 HalfCheetah-v2

```
<mujoco model="cheetah">
  <compiler angle="radian" coordinate="local" inertiafromgeom="
     true" settotalmass="14"/>
  <default>
    <joint armature=".1" damping=".01" limited="true"
       solimplimit="0 .8 .03" solreflimit=".02 1" stiffness="8"/
       >
    <geom conaffinity="0" condim="3" contype="1" friction=".4 .1
        .1" rgba="0.8 0.6 .4 1" solimp="0.0 0.8 0.01" solref="
       0.02 1"/>
    <motor ctrllimited="true" ctrlrange="-1 1"/>
  </default>
  <size nstack="300000" nuser_geom="1"/>
  <option gravity="0 0 -9.81" timestep="0.01"/>
  <asset>
    <texture builtin="gradient" height="100" rgb1="1 1 1" rgb2="
       0 0 0" type="skybox" width="100"/>
    <texture builtin="flat" height="1278" mark="cross" markrgb="
       1 1 1" name="texgeom" random="0.01" rgb1="0.8 0.6 0.4"
       rgb2="0.8 0.6 0.4" type="cube" width="127"/>
    <texture builtin="checker" height="100" name="texplane" rgb1
       ="0 0 0" rgb2="0.8 0.8 0.8" type="2d" width="100"/>
    <material name="MatPlane" reflectance="0.5" shininess="1"
       specular="1" texrepeat="60 60" texture="texplane"/>
    <material name="geom" texture="texgeom" texuniform="true"/>
```

```
</asset>
<worldbody>
  <light cutoff="100" diffuse="1 1 1" dir="-0 0 -1.3"
     directional="true" exponent="1" pos="0 0 1.3" specular="
     .1 .1 .1"/>
  <geom conaffinity="1" condim="3" material="MatPlane" name="
     floor" pos="0 0 0" rgba="0.8 0.9 0.8 1" size="40 40 40"
     type="plane"/>
  <body name="torso" pos="0 0 .7">
    <camera name="track" mode="trackcom" pos="0 -3 0.3" xyaxes
       ="1 0 0 0 0 1"/>
    <joint armature="0" axis="1 0 0" damping="0" limited="
       false" name="rootx" pos="0 0 0" stiffness="0" type="
       slide"/>
    <joint armature="0" axis="0 0 1" damping="0" limited="
       false" name="rootz" pos="0 0 0" stiffness="0" type="
       slide"/>
    <joint armature="0" axis="0 1 0" damping="0" limited="
       false" name="rooty" pos="0 0 0" stiffness="0" type="
       hinge"/>
    <geom fromto="-.5 0 0 .5 0 0" name="torso" size="0.046"
       type="capsule"/>
    <geom axisangle="0 1 0 .87" name="head" pos=".6 0 .1" size
       ="0.046 .15" type="capsule"/>
    <!-- <site name='tip' pos='.15 0 .11'/>-->
    <body name="bthigh" pos="-.5 0 0">
      <joint axis="0 1 0" damping="6" name="bthigh" pos="0 0 0"
          range="-.52 1.05" stiffness="240" type="hinge"/>
      <geom axisangle="0 1 0 -3.8" name="bthigh" pos=".1 0 -.13
         " size="0.046 .145" type="capsule"/>
      <body name="bshin" pos=".16 0 -.25">
        <joint axis="0 1 0" damping="4.5" name="bshin" pos="0 0
            0" range="-.785 .785" stiffness="180" type="hinge"/
           >
        <geom axisangle="0 1 0 -2.03" name="bshin" pos="-.14 0
           -.07" rgba="0.9 0.6 0.6 1" size="0.046 .15" type="
           capsule"/>
        <body name="bfoot" pos="-.28 0 -.14">
          <joint axis="0 1 0" damping="3" name="bfoot" pos="0 0
              0" range="-.4 .785" stiffness="120" type="hinge"/
             >
          <geom axisangle="0 1 0 -.27" name="bfoot" pos=".03 0
             -.097" rgba="0.9 0.6 0.6 1" size="0.046 .094" type
             ="capsule"/>
        </body>
```

```
      </body>
    </body>
    <body name="fthigh" pos=".5 0 0">
      <joint axis="0 1 0" damping="4.5" name="fthigh" pos="0 0
         0" range="-1 .7" stiffness="180" type="hinge"/>
      <geom axisangle="0 1 0 .52" name="fthigh" pos="-.07 0
         -.12" size="0.046 .133" type="capsule"/>
      <body name="fshin" pos="-.14 0 -.24">
        <joint axis="0 1 0" damping="3" name="fshin" pos="0 0 0
           " range="-1.2 .87" stiffness="120" type="hinge"/>
        <geom axisangle="0 1 0 -.6" name="fshin" pos=".065 0
           -.09" rgba="0.9 0.6 0.6 1" size="0.046 .106" type="
           capsule"/>
        <body name="ffoot" pos=".13 0 -.18">
          <joint axis="0 1 0" damping="1.5" name="ffoot" pos="0
              0 0" range="-.5 .5" stiffness="60" type="hinge"/>
          <geom axisangle="0 1 0 -.6" name="ffoot" pos=".045 0
             -.07" rgba="0.9 0.6 0.6 1" size="0.046 .07" type="
             capsule"/>
        </body>
      </body>
    </body>
  </worldbody>
  <actuator>
    <motor gear="120" joint="bthigh" name="bthigh"/>
    <motor gear="90" joint="bshin" name="bshin"/>
    <motor gear="60" joint="bfoot" name="bfoot"/>
    <motor gear="120" joint="fthigh" name="fthigh"/>
    <motor gear="60" joint="fshin" name="fshin"/>
    <motor gear="30" joint="ffoot" name="ffoot"/>
  </actuator>
</mujoco>
```

B.2 Walker2D-v2

```
<mujoco model="walker2d">
  <compiler angle="degree" coordinate="global" inertiafromgeom="
     true"/>
  <default>
    <joint armature="0.01" damping=".1" limited="true"/>
    <geom conaffinity="0" condim="3" contype="1" density="1000"
       friction=".7 .1 .1" rgba="0.8 0.6 .4 1"/>
```

```
</default>
<option integrator="RK4" timestep="0.002"/>
<worldbody>
  <light cutoff="100" diffuse="1 1 1" dir="-0 0 -1.3"
     directional="true" exponent="1" pos="0 0 1.3" specular="
     .1 .1 .1"/>
  <geom conaffinity="1" condim="3" name="floor" pos="0 0 0"
     rgba="0.8 0.9 0.8 1" size="40 40 40" type="plane"
     material="MatPlane"/>
  <body name="torso" pos="0 0 1.25">
    <camera name="track" mode="trackcom" pos="0 -3 1" xyaxes="
       1 0 0 0 0 1"/>
    <joint armature="0" axis="1 0 0" damping="0" limited="
       false" name="rootx" pos="0 0 0" stiffness="0" type="
       slide"/>
    <joint armature="0" axis="0 0 1" damping="0" limited="
       false" name="rootz" pos="0 0 0" ref="1.25" stiffness="0
       " type="slide"/>
    <joint armature="0" axis="0 1 0" damping="0" limited="
       false" name="rooty" pos="0 0 1.25" stiffness="0" type="
       hinge"/>
    <geom friction="0.9" fromto="0 0 1.45 0 0 1.05" name="
       torso_geom" size="0.05" type="capsule"/>
    <body name="thigh" pos="0 0 1.05">
      <joint axis="0 -1 0" name="thigh_joint" pos="0 0 1.05"
         range="-150 0" type="hinge"/>
      <geom friction="0.9" fromto="0 0 1.05 0 0 0.6" name="
         thigh_geom" size="0.05" type="capsule"/>
      <body name="leg" pos="0 0 0.35">
        <joint axis="0 -1 0" name="leg_joint" pos="0 0 0.6"
           range="-150 0" type="hinge"/>
        <geom friction="0.9" fromto="0 0 0.6 0 0 0.1" name="
           leg_geom" size="0.04" type="capsule"/>
        <body name="foot" pos="0.2 0 0">
          <joint axis="0 -1 0" name="foot_joint" pos="0 0 0.1"
             range="-45 45" type="hinge"/>
          <geom friction="0.9" fromto="-0.0 0 0.1 0.2 0 0.1"
             name="foot_geom" size="0.06" type="capsule"/>
        </body>
      </body>
    </body>
    <!-- copied and then replace thigh->thigh_left, leg->
       leg_left, foot->foot_right -->
    <body name="thigh_left" pos="0 0 1.05">
      <joint axis="0 -1 0" name="thigh_left_joint" pos="0 0
```

```
        1.05" range="-150 0" type="hinge"/>
      <geom friction="0.9" fromto="0 0 1.05 0 0 0.6" name="
        thigh_left_geom" rgba=".7 .3 .6 1" size="0.05" type="
        capsule"/>
      <body name="leg_left" pos="0 0 0.35">
       <joint axis="0 -1 0" name="leg_left_joint" pos="0 0 0.6
          " range="-150 0" type="hinge"/>
       <geom friction="0.9" fromto="0 0 0.6 0 0 0.1" name="
          leg_left_geom" rgba=".7 .3 .6 1" size="0.04" type="
          capsule"/>
       <body name="foot_left" pos="0.2 0 0">
        <joint axis="0 -1 0" name="foot_left_joint" pos="0 0
           0.1" range="-45 45" type="hinge"/>
        <geom friction="1.9" fromto="-0.0 0 0.1 0.2 0 0.1"
           name="foot_left_geom" rgba=".7 .3 .6 1" size="0.06
           " type="capsule"/>
       </body>
      </body>
    </body>
  </body>
</worldbody>
<actuator>
  <!-- <motor joint="torso_joint" ctrlrange="-100.0 100.0"
     isctrllimited="true"/>-->
  <motor ctrllimited="true" ctrlrange="-1.0 1.0" gear="100"
     joint="thigh_joint"/>
  <motor ctrllimited="true" ctrlrange="-1.0 1.0" gear="100"
     joint="leg_joint"/>
  <motor ctrllimited="true" ctrlrange="-1.0 1.0" gear="100"
     joint="foot_joint"/>
  <motor ctrllimited="true" ctrlrange="-1.0 1.0" gear="100"
     joint="thigh_left_joint"/>
  <motor ctrllimited="true" ctrlrange="-1.0 1.0" gear="100"
     joint="leg_left_joint"/>
  <motor ctrllimited="true" ctrlrange="-1.0 1.0" gear="100"
     joint="foot_left_joint"/>
  <!-- <motor joint="finger2_rot" ctrlrange="-20.0 20.0"
     isctrllimited="true"/>-->
</actuator>
  <asset>
      <texture type="skybox" builtin="gradient" rgb1=".4 .5 .6"
          rgb2="0 0 0"
         width="100" height="100"/>
      <texture builtin="flat" height="1278" mark="cross"
         markrgb="1 1 1" name="texgeom" random="0.01" rgb1="
```

```
            0.8 0.6 0.4" rgb2="0.8 0.6 0.4" type="cube" width="
            127"/>
        <texture builtin="checker" height="100" name="texplane"
            rgb1="0 0 0" rgb2="0.8 0.8 0.8" type="2d" width="100"
            />
        <material name="MatPlane" reflectance="0.5" shininess="1"
             specular="1" texrepeat="60 60" texture="texplane"/>
        <material name="geom" texture="texgeom" texuniform="true"
            />
    </asset>
</mujoco>
```