Explainable Reinforcement Learning

Discovering intent based explanations for heterogeneous cooperative multi agent reinforcement learning agents

Ada Hatland

Informatics: Robotics and Intelligent Systems
60 ECTS study points

Department of Informatics
The Faculty of Mathematics and Natural Sciences

Spring 2025

Ada Hatland

Explainable Reinforcement Learning

Discovering intent based explanations for heterogeneous cooperative multi agent reinforcement learning agents

Supervisors:
Dr. Dennis Groß
Prof. Kyrre Glette
Dr. Helge Spieker

Declaration of AI use

In this thesis generative models have been used for topic suggestion, a small part of figure creation, equations and parts of the code used for creating plots, as well as debugging and documentation. All code suggested has been verified by a human. Generative models have **not** been used for text generation. I, the author, take full responsibility for the contents of this thesis.

Acknowledgments

I would like to thank my external supervisor Dennis Groß, as well as my internal supervisor Kyrre Glette for their guidance and support in writing this thesis. They have been of significant help when discussing how to proceed and what remains on the to-do list. I would also like to thank my cat Nokia for emotional support throughout the writing of this thesis.

List of Acronyms

Abstract

This is the abstract

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Sequential decision-making problems are problems where one decision leads to a new state that requires a new decision to be made. An example of this is autonomously driving from point A to point B through city streets. For these types of problems ***rl! (rl!)*** systems are used. **rl!**, both multi- and single-agent models have seen a significant rise in successful use and applicability in recent years, with models such as AlphaGo[**article**] and smacv2[**ellis2023smacv2**]. These models learn by agents performing actions in a given state decided by a policy that lead to a new state, and learning by receiving rewards depending on if the new state is preferable to the previous, the aim is to learn a near optimal policy for achieving a fixed goal[**Sutton1998**]. The agent uses an observation of the state to choose an action. The environment describes how an action affects the state. See Figure **??**. In a ***marl! (marl!)*** system, we would have multiple agents.

1.1 Motivation

A significant problem with many deep machine learning models, including **marl!** policies, is known as the black box problem[**zednik2019solving**]. This problem describes how the processing of information is hidden from a human user due to the opacity of the model, and we just have to trust that the model uses relevant data to come to the conclusion it does, which it often doesn't do. For many tasks and models where the impact of the output isn't highly significant this isn't a big issue, but for tasks like autonomous driving we simply cannot use these models without trust for the models processing of data and that the solution given wont hurt anyone. We can considering autonomous driving as a **marl!** system if we consider each vehicle as an agent. To solve this we would like to have models that along with an output can provide us with some kind of reasoning for what information is used and how.

All this essentially means, until the black box problem is solved, autopilot will always require a driver behind the wheel[**tian2018deeptest**]. Despite having high accuracy, precision and recall, a reinforcement learning model might choose a less than preferable action in edge cases or states not well covered by training and test sets, for example driving on snowy roads when all images in the training data is from warmer climates, or combinations that aren't well covered, like a foggy, snowy road with a sharp right turn. With a way to ask the agent for intent, we could find that it has learned to always expect there to not be a car around the corner, which is not always obvious from looking at the dataset, but if it in a decision relies on the road to be empty to choose a safe route, this is obviously a huge issue. This paper aims to explain how an agent in a **marl!** setting decides on an action due to what it expects from other agents in the future. The field working on combating this issue is known is Explainable reinforcement learning.

1.2 Problem statement

The focus of this paper will be to rework and apply methods for general ***xai! (xai!)*** to Reinforcement Learning, and expand on ***xrl! (xrl!)*** methods already developed. In particular we will focus on an agent and how the expected future states of other agents will affect it in its decision making process. The environment we will focus on is a **marl!** environment known as Knights Archers Zombies [**KAZ**] made for Python. We use this because it is a cooperative environment where each agent has to consider other agents, both of the same type as itself, and other types. If we used an environment where all the agents are identical, our findings will be less useful for other environments where the agents aren't identical. In many real life scenarios agents will not be identical. If we once again consider autonomous vehicles, most vehicles are different in some way. A very obvious example of this is considering cars and motorcycles. Where because of their size difference, their paths chosen will often be different.

In psychology it is well known that most human based decisions are made with intent[**inbook**], and by focusing on the expectation of what future states will look like we can consider this the intent of the agent. If we are accurately able to extract the intent of an **rl!** policy we are better able to explain the choice made and we are more comfortable with trusting that the choice made is a good decision. Concretely we hypothesise that by analyzing expected future states and events we could find the intent of the agent, which we could use to ensure the agent acts in a safe or desirable manner. The main question we aim to answer is what information, found in weights or design of the policy, or information extracted by knowing the weights and design, is the most important for separate prediction models to be able to predict future states and actions, in cases where we do not have access to a decent world model, or such a model does not exist.

1.3 Scope and limitations

We aim to construct a framework to describe the intent of the agents in the KAZ environment, and we aim to make it applicable to other environments as well. However, due to time limitations we restrict the framework to only accept PettingZoo example environments or environments created with the PettingZoo environment creation tool. **marl!** environments can be divided into two major categories, ***aec! (aec!)*** and parallel. **aec!** environments are environments where the state transition happens after each agent chooses an action, and parallel environments are environments where all the agents choose an action each before the transition happens. See figure **??**.

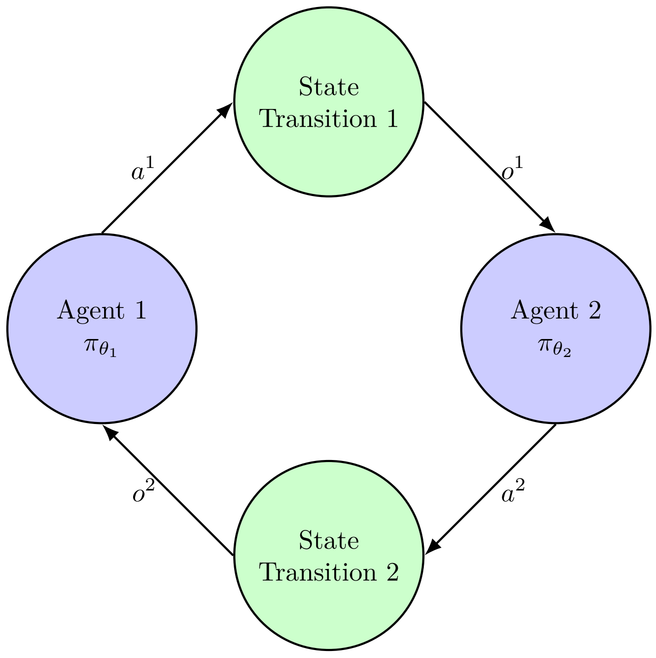


Figure 1.1: Example **aec!** in an environment with 2 agents.

Due to limited time and resources the size of policy or value networks, as well as any other ***nn!**s (**nn!**s)* or otherwise computationally expensive algorithm used, will have to be significantly limited, this will impact results in a meaningful way, and therefore we will keep all **nn!**s comparable sizes, and focus on doing experiments on how other factors than network size impact the results.

We do not have access to any real world datasets, so its hard to ensure the methods we have developed will be directly applicable, however the results we get will still be relevant for later research in the area.

1.4 Research methods

This section describes the methods we use to answer the questions we have posed. The methodology mainly focuses on produced datasets made with a digital **rl!** environment, which does mean we have access to a world model, however as we want to answer the research questions without the use of a world model we will only be using the results of simulations to create our methods. The effect of this is that the methods will be compatible with **rl!** systems where we do not have access to a world model.

Producing our own dataset has the benefit of no restrictions, other than hardware restrictions, on the datasets size. We can also produce variations of these datasets by altering the simulations used to create them which would in a real world setting often be impossible, e.g. access to gradients during inference.

The datasets produced will be sets of full or partial trajectories in given environments along with other relevant data like integrated gradient, or Shapley values for the observations in the trajectories. We use these datasets to train predictive **nn!**s with two main types. Event prediction, for example whether an agent encounters a critical state within the next 10 timesteps, which could represent a dangerous situation for an autonomous vehicle, and state prediction, which for example could be the x and y coordinates of a given agent 10 timesteps into the future.

We will first be focusing on including explanations for the observations and observe and analyze the effects on event and state predictors, then we will be using **nn!** policies with access to memory and include their representation of memory along with the observation, hidden values for lstm and once again observe and analyze effect on the predictors.

1.5 Ethical considerations

marl! is an important field in development for military purposes[**military_marl**], and while this research is not supposed to effectivize warfare its possible that it ends up being relevant for unintended areas of research, including but not limited to military use-cases. This is especially the case because I aim to develop methods for general **marl!** use and not a specific area of research, like medicine or sports. It is more or less impossible to restrict access to these methods for specific areas of research.

If this research aids to make **marl!** methods applicable to real world settings there is the possibility it risks increasing job displacement issues. This is an issue that affects most ***ai! (ai!)*** research and can be mitigated by programs to reeducate people whose jobs it might affect, and increasing employment in developing and monitoring **ai!** tools.

1.6 Main contributions

As discussed in **??**, **??** and **??**, there are certain specific objectives we aim to reach. In general we aim to expand on current future prediction research.

- **Objective 1: Observation space**
  Most current methods assumes the observation space to be image based, and bases research on information found in convolutional layers, either regular ***cnn!s (cnn!s)***, or convLSTM layers.
- **Objective 2: Feature importance as input**
  Using feature importance from the policy to improve future predictions is a novel idea, and has potential to improve future predictions by a significant margin.
- **Objective 3: Memory as input**
  Earlier research has found that including hidden states from the ***drc! (drc!)*** architecture had a significant effect on accuracy when predicting future states and events. We aim to expand on this and use hidden states from ***lstm!s (lstm!s)*** layers and observe the effect and analyze how it differs from using the ConvLSTM hidden values.

1.7 Thesis outline

This thesis will be structured as follows:

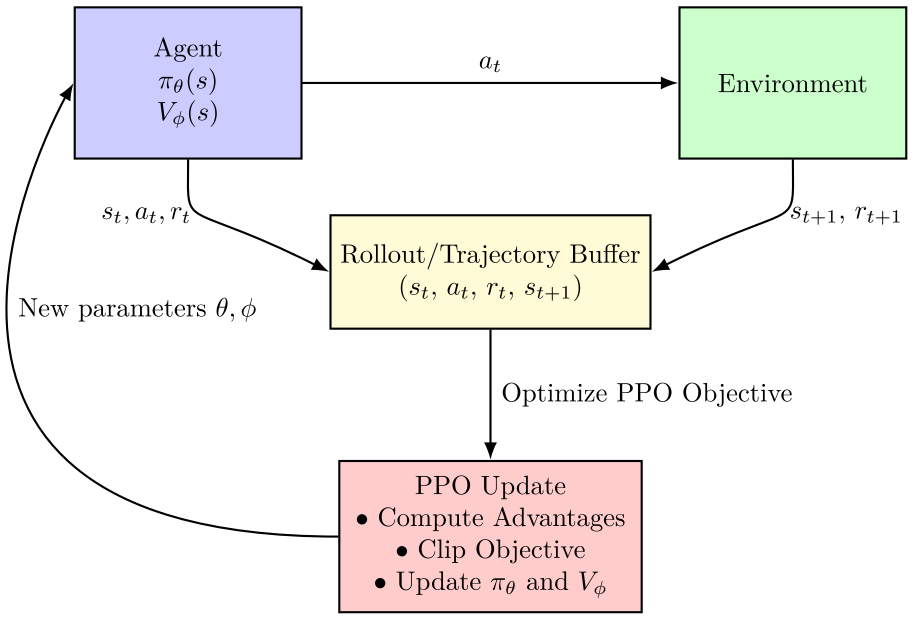


Figure 1.2: Diagram showing the ***ppo! (ppo!)*** update algorithm

- **Chapter 2: Background and related works**
  This chapter will introduce important concepts, both **rl!** and explainability related, and briefly discuss the history of **xai!** methods, and how they are applicable to my use case. We will also consider **xrl!** methods that are already developed and how to expand or integrate them into developed methods of our own.
- **Chapter 3: Methodology**
  This chapter discusses the specific principles, procedures and techniques used to conduct the **xrl!** research done in this thesis, to ensure that the results we get are valid and reliable.
- **Chapter 4: Experiments**
  This chapter details the setup, results and analysis of the specific experiments we use to evaluate the hypothesis and answer research questions based on certain evaluation metrics, and baseline comparisons.
- **Chapter 5: Discussion**
  This chapter will analyze the results of the experiments and discuss their implication for **xrl!** research. It will also contain the limitations of our work and briefly discuss how to expand on any research done.

Chapter 2

Background and Related Works

2.1 Reinforcement Learning Fundamentals

To make sure all methods are understood well its important to make sure we have a proper understanding of the fundamentals of **rl!**, especially ***drl! (drl!)*** and **marl!**. This section will go over the basics needed.

2.1.1 Markov Decision Process in Reinforcement Learning

A ***mdp! (mdp!)*** is a framework used to model decision-making in stochastic environments, like we want to in an **rl!** task. It is defined by a tuple with 5 elements:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the set of possible actions, $P(s'|s, a)$ is the transition probability function, which defines the probability of moving to state s' when action a is taken in state s, $R(s, a, s')$ is the reward function that provides a reward for transitioning from state s to s' via action a, $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards.

The objective in an **mdp!** is to find a policy $\pi(s)$ that maximizes the expected cumulative reward:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \mid \pi\right]$$

where $V^{\pi}(s)$ is the value function that represents the expected return starting from state s and following policy π. See figure **??**.

2.1.2 Markov Decision Process in Multi Agent Reinforcement Learning

In a **marl!** setting agents could have shared or independent **mdp!**s depending on architecture. There are three main branches of **marl!**, cooperative, competitive and mixed. Our focus will be on cooperative **marl!**. In a cooperative **marl!** setting the goal is some social welfare function that maximises rewards for the agents, either collective rewards or a mix of collective and individual rewards. In such a setting each agent has **mdp!**.

Assuming that state space $\mathcal{S}$ and discount factor γ is shared, which they will be in all my experiments, if two agents have the same reward structure R and action space $\mathcal{A}$ their objective would be the same, in which case they could share policy π and could therefore share $\mathcal{M}$.

2.1.3 Deep Reinforcement Learning

Traditional RL methods struggle with scalability due to the fact that they rely on discrete state representations. **drl!** uses deep **nn!**s to approximate the policy function $\pi_\theta(s)$, the value function $V_\phi(s)$, or the Q-function $Q_\theta(s,a)$, making it possible to represent these functions in continuous and complex environments. See figure **??**. Input layer is size of state for all three functions. Output layer is size of action space for $Q_\theta(s,a)$ and $\pi_\theta(s)$. $V_\phi(s)$ only has one output node. In a feed forward **nn!** like we have here each node has the value of some activation function $\phi(x)$ where x is the sum of nodes of the previous layer multiplied by their respective weights, usually as well as adding some bias, shown as the connections between the nodes. Common activation functions are $ReLU(x) = max(0,v)$ and $tanh = \frac{e^v - e^v}{e^v + e^v}$ where v is the value before activation.

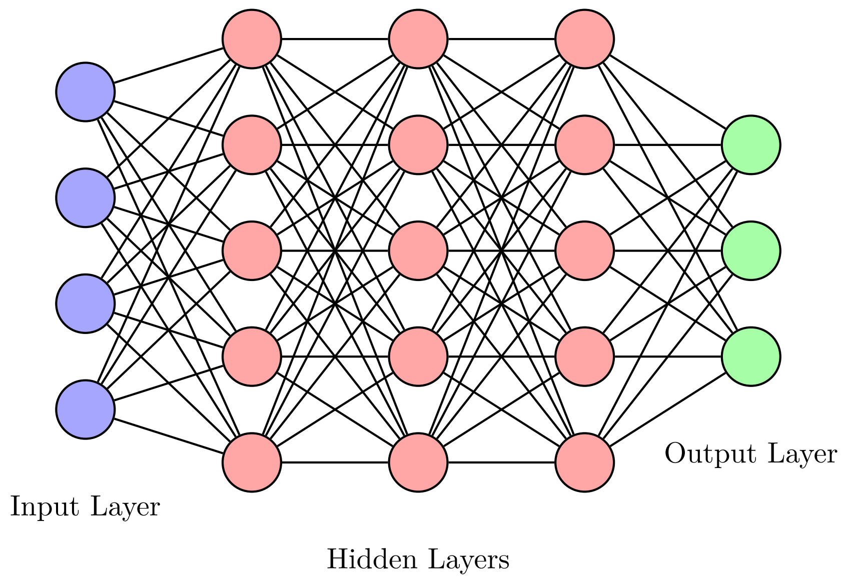


Figure 2.1: Illustration of an example of a **nn!** with three hidden layers

2.1.4 Model-Free vs. Model-Based Reinforcement Learning

Model-based and model-free reinforcement learning differ in how they represent environment transitions. In model-based reinforcement learning, the agent learns or has access to a model of the environment, which includes a transition function $P(s'|s,a)$ and a reward function $R(s,a)$ that determines the expected reward for taking action a in state s [**moerland2022modelbasedreinforcementlearningsurvey**]. This allows the agent to simulate future outcomes without taking actions in the real environment. As a result, model-based methods are

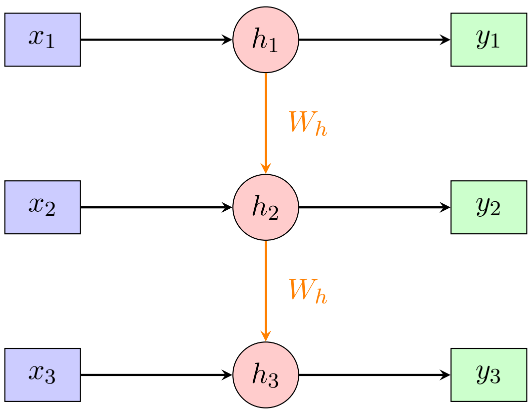


Figure 2.2: Recurrent neural network showing a 3 step time series with hidden values propagating through time.

able to update policies based on imagined rollouts, which increases learning speed compared to executing the real environment transitions. Examples of model-based algorithms are Dyna[**10.1145/122344.122377**] and DreamerV3[**hafner2024masteringdiversedomainsworld**].

In contrast, model-free reinforcement learning does not learn or utilize an explicit model of the environment. Instead, the agent learns by interacting directly with the environment and updating its network weights based on observed rewards. This approach is generally less sample-efficient but is applicable to environments where modeling the transition dynamics isn't feasible. Examples of model-free algorithms include Q-learning, ***dqn! (dqn!)*** [**mnih2013playingataridee preinforcement**], and **ppo!** [**schulman2017proximalpolicyoptimizationalgorithms**].

Model-based methods enable explicit planning by using the learned environment transitions in methods such as Monte Carlo Tree Search [**__wiechowski__2022**], while model-free methods rely on direct experience to optimize behavior. We will be using **ppo!** to optimize our algorithms, and will not be using simulations or any other form of explicit planning, ensuring that the methods developed will be applicable in the most amount of environments.

2.1.5 Recurrency in Deep Reinforcement Learning Policies

Recurrency in **nn!** is a way to carry over information from previous timesteps to current ones, if we believe that information could be useful. ***rnn!****s* ***(rnn!****s)* are sometimes used when constructing the architecture of **drl!** policies [**hausknecht2017deeprecurrentqlearningpartially**], mainly for two reasons. The first one is for when your environment can be modeled as a ***pomdp! (pomdp!)***, in which case memory is important as the current observation is not always enough to capture all the information an agent could have. The second reason is for planning, like in the **drc!** architecture[**guez2019investigationmodelfreeplanning**]. See figure **??**.

2.1.6 Backpropagation and gradient descent

Backpropagation is a training algorithm used to train ***dnn!s (dnn!s)*** to minimize error. If a network is differentiable you can use backpropagation. First you do a forward pass through the network, i.e. your network calculates a prediction based on an input. Then you calculate loss, the difference between true value and predicted value is passed to some function. MSE is one such function.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where y is true value, $\hat{y}$ is predicted value, n is amount of samples multiplied by outputs. If you are trying to predict (x, y) coordinates, you would have two outputs, if you're trying to predict x, y coordinates in 5 cases you would have $n = 10$. Then you do a backwards pass, calculating loss with respect to the weights in the network using the chain rule. Then using gradient descent you change the weights to reduce loss.

2.2 Approaches to Explainability in Reinforcement Learning

2.2.1 Post Hoc vs. Intrinsically Explainable Models

This section will discuss benefits and drawbacks of explainable models divided into two main categories, Post hoc models and intrinsically explainable models.

Post hoc explainability refers to finding ways to understand already trained models. Instances where this doesn't pose more of a challenge can often be more efficient as you do not need to construct and train a model, which could often pose an issue both considering time spent and accuracy of the model. Making a model explainable only makes sense if we know it usually makes sensible decisions. There are however issues with making post hoc models.

Temporal Aspect

rl! policies all have a temporal aspect, which means several different actions over a certain period of time might contribute to a singular outcome, this could make it very challenging to pinpoint which actions were made for which outcome, especially if the outcome happens several states after the initial decision was made.

Nature of Black Box Models

Many **rl!** policies, especially deep **rl!** policies, which are the policies we will be working with, have complex inner connections due to the hierarchical structure of learning features in the input, sequences of non linear connections that are very hard to understand without spending a lot of time studying the specific connections learned by a model. It's often very difficult to accurately pinpoint the purpose of a node, especially because we do not know if it even has a purpose at all or it could have a lot of different purposes. Especially in environments with complex observation spaces.

Lack of Transparency

Tracing how an observation leads to a state, through the action chosen and the environment, is often challenging in models constructed without this in mind. If we do not know the processing of information of a model it's very hard to explain the rationalisation of an agent.

Intrinsically Explainable Models

Intrinsically explainable models have the already stated drawback of relying on careful model construction, which can be challenging and may impact model performance. While these models offer transparency from the get go, they are often less flexible and might not achieve the same level of accuracy as more complex **rl!** models. Given that we aim to analyze and compare already trained **rl!** policies, we will mostly focus on post hoc explainability methods in this thesis.

2.2.2 Fidelity of Explainable Artificial Intelligence Methods

It is important to understand why the current **xai!** methods not meant for **rl!** applications aren't always useful to us.

xai! methods not intended for reinforcement learning often provide explanations that don't necessarily represent the inner workings of an **rl!** policy, because an **rl!** policy has a temporal aspect as well. Broadly these methods can be categorised as feature-based explainers, and they often struggle to fully explain an agents behaviour or specific actions because they cannot capture the future view of an **rl!** policy.

Saliency maps which have been successfully used for classification of images provide explanations about what part of the input was important for the outcome, which is highly relevant for classification tasks, but using the same method for an **rl!** policy doesn't sufficiently explain the intent of the agent[**atrey2020exploratory**].

Another commonly used **xai!** method is model distillation, which works by transferring knowledge from a large model to a smaller, usually interpretable, one, for instance a deep learning network to a decision tree [**bastani2019verifiable**]. This has use cases in verifyability, but struggles to fully explain the temporal aspect of **rl!** policies, and are therefore not sufficient as an explainer.

However, these methods might still prove insightful in conjunction with other intent-based methods, the state in which a decision is made is obviously very relevant to why that particular decision was made. We could perhaps use these methods to answer questions such as "What part of agent As Observation this state, lead it to believe agent B would end up at these coordinates at a later state?"

2.2.3 Future-Based Explanations

Next, I will describe in slightly more detail, what is meant by an intent based explainer, like I want to develop, and how to use it.

Design

"Towards Future-Based Explanations for Deep RL Network Controllers"[**10.1145/3626570.36266** broadly describes future-based intrinsically explainable methods. Future-based intrinsically explainable methods for **drl!** policies often take three inputs, the trajectories experienced by an agent, the environment and the agent. Then, they collect the rewards and interactions and use this information to train an explainer.

During inference, we can then apply the explainer to a state and action to get the expected future consequences of that action. Depending on the architecture we could either get the expected future consequence of any action, or just the one the agent decides on.

Use Cases

Designing a **drl!** solution requires choosing features in the observation, hyperparameter tuning, policy design and reward function among other things. This is a time, and resource, consuming process. These are usually picked by trial-and-error but could be made easier with assistance from an explainer.

Another, and perhaps more important, use case is safety. If when online, i.e. the chosen action will actually affect the state, we are expecting high likelihood of an unsafe state, we could instead of opting for the chosen action fall back to a known safe action, that could have a lower expected return, i.e. breaking instead of turning a corner if we have limited vision around the corner.

Definition of Intent

We define the state transition function $T(s, \mathbf{a}, s')$ where $\mathbf{a}$ is the set of simultaneous actions made by the set of active agents, in our case, one action per agent. Given T the agent should when prompted output a set of trajectories $\tau = \{(s_0, \mathbf{a}_0), (s, \mathbf{a}), (s_1, \mathbf{a}_1)...(s_n, \mathbf{a}_n)\}$ where s_0 is the current state, $\mathbf{a}_i$ is the set of actions taken in s_i, and s_{i+1} is the state reached by $T(s_i, \mathbf{a}_i, s_{i+1})$. We will apply a series of methods on τ to extract intent. One of these methods is discovering counterfactual trajectories, $'\tau$, where the actions made by the agents in $'\tau$ are as similar as possible to the actions made by the agents in τ, but the total reward gained is as low as possible. $'s_0 = s_0$ and given an identical transition function T the goal is to discover which actions are most important to receive the reward. This method is similar to and inspired by ACTER[**gajcin2024acter**], see section **??**. Details in section **??**.

2.3 Explainable Deep Reinforcement Learning: State of the Art and Challenges

This paper is a comprehensive survey of the most common state of the art methods applicable to **rl!**. Section 4 specifically highlights ***xdrl! (xdrl!)*** methods. This subsection will focus on these methods[**sota**]. This section will serve to give an overview of **xrl!** methods that already exist, and if and how I use these methods to develop my own.

2.3.1 Feature importance

LIME and SP-LIME

LIME and SP-LIME are model-agnostic methods that explain chosen actions in specific states by slightly modifying input features and fitting an interpretable linear surrogate model that approximates the behaviour of the model locally. SP-LIME selects a representative set of these explanations to create a global explanation for how the model selects an action. These methods explain what features are important when deciding on an action in a given state.

Shapley Additive exPlanations

Shapley values is a method of computing marginal contributions from cooperative game theory. It assigns a Shapley value to each feature which is the mean of that features marginal contribution over all combinations of features to establish a fair attribution of credit to each player, or feature in the case of machine learning. This method is very computationally expensive and as such, approximations of this method are used instead.

Layer-wise Relevancy Propagation

LRP is a method to assign importance to each node in the network by working backwards from the output to the input. It looks at what nodes in the last hidden layer influenced the output and redistributes prediction scores to these nodes, it then looks at what nodes in the layer before influenced the nodes in the last hidden layer, until you have a complete map of how the network nodes affected each other to produce the output. This method helps reveal hidden patterns in model behaviour.

2.3.2 Policy explanations

HIGHLIGHTS

This is a intrinsically explainable **rl!** algorithm and it works by extracting sub-trajectories with important states, states where a wrong action can lead to significant decreases in reward, to explain how the agent acts in these states. It also provides context in the form of neighbouring states in the important sub-trajectories, in order for users to understand why it chose the trajectory it did.

2.3.3 Objective explanations

2.3.4 Outcome explanation

2.4 Relevant Methods

There are several papers written on **xai!** and **xrl!** problems. Milani et al.[**milani2022survey**] categorise **xrl!** into three main categories, feature importance (FI), learning process and **mdp!** (LPM) and Policy-level (PL). FI considers the immediate context for actions, i.e. what part of the input was important for a single action, LPM considers techniques that highlight which experiences in training were influential for the final policy and PL focuses on long term effects and behaviour. Since we are interested in future states and actions we will look at influential trajectories and transitions within these trajectories. It is important to view these transitions in the context of the trajectory to understand the long term effects and not just immediate, which are of less interest in this paper, if we find the state with a high state importance $I(s)$, $I(s) = max_a Q(s,a) - min_a Q(s,a)$ most similar by some similarity measure to an arbitrary current state we could find the resulting trajectory and expect the agent to intend a similar outcome. There are also ways to convert **rnn!** policies to an interpretable models post-hoc, which might be relevant if we use an **rnn!**. This paper will explore PL explainability further.

In particular "What did you think would happen? Explaining Agent Behaviour through Intended Outcomes" [**yau2020did**], "Explainable Reinforcement Learning via a Causal world model" [**yu2024explainable**] and "CrystalBox: Future-based explanations for input-driven deep **rl!** systems" [**patel2024crystalbox**] are highly relevant due to the fact that they all describe temporal connections between current actions and future states or actions.

"Are large language models post-hoc explainers" [**kroeger2024large**] Could be relevant as using other explainers to compare explanations is useful. "ACTER: Diverse and Actionable Counterfactual Sequences for Explaining and Diagnosing **rl!** Policies" [**gajcin2024acter**]

2.4.1 What did you think would happen? Explaining Agent Behaviour through Intended Outcomes

What did you think would happen describes what an agent expects to happen in future states, and why the current action is chosen based on the future expectations. As stated in the paper, a limitation of their method means it doesn't work well with high dimensionality. The two main difference between this paper and the problem i aim to solve is that they're focusing on an environment with a single agent and that our observation space will be high dimensionality. It uses Q-learning and Markov-chains that train simultaneously with a "belief map" that shows what the agent expects the environment to look like in future states. In the simple examples used in the paper it shows where it believes the taxi should drive and therefore chooses an action to follow this path. This is not directly applicable to my thesis as it's unlikely that Q-learning or Markov chains will be viable for the policy and explainer based on the environment. However the paper is successful in explaining an agents underlying motivations and beliefs about the causal nature of the environment, and using similar methods might be an effective means for making **marl!** agents with higher dimensionality understandable from a human perspective.

2.4.2 Explainable Reinforcement Learning via a Causal world model

Explainable Reinforcement Learning via a Causal world model constructs a sparse model to connect causal relationships, without prior knowledge of the causal relationship, rather than a fully connected one, but they still achieve high accuracy and results comparable to other fully connected ***mbrl! (mbrl!)*** policies. Which is important, as there is often a trade off between explainability and performance. Using the same model for both explanations and choosing actions also make the explanations faithful to the intentions of the agent. The paper also describes a novel approach to derive causal chains from the causal influence of the actions, which lets us find the intent of the agent. The paper is successful being applied to **mbrl!**, and is also applicable to models with infinite actions spaces, which is a limitation of some other models, see previous sub section.

A limitation of the paper is that it requires a known factorisation of the environment, denoted by $\langle S, A, O, R, P, T, \gamma \rangle$, where S is state space, A is action space, O is observation space, R is reward function, P is probability function for the probability of transitioning from state s to state s' given action a, T is the termination condition given the transition tuple (s, a, o, s'), and γ is the discount factor. Considering we will be working with hand crafted simulations we will have access to all of these, however its not certain that if we depend on this method that our contributions will be applicable to certain other environments where the factorisation is not known.

2.4.3 CrystalBox: Future-Based Explanations for Input-Driven Deep Reinforcement Learning Systems

CrystalBox introduces a model-agnostic, post-hoc, future based explanation for **drl!**. It doesn't require altering the controller, and works by decomposing the future rewards into its individual components. While this isn't exactly the kind of explanation we are looking for, it could be a great tool in developing an explainer that considers other agents actions in a multi agent cooperative environment, which is the goal of our paper, because it is post-hoc, and easily deployable. Especially because it was constructed for input-driven environments. The original paper claims it offers high fidelity solutions for both discrete and continuous action spaces. KAZ has a discrete action space but we might do some work with other environments as well.

It's not certain to be useful because it works by decomposing the reward function, and it's not safe to assume the reward function will even be useful to decompose.

2.4.4 Are Large Language Models Post Hoc Explainers

A ***llm! (llm!)*** is a predictive model that generates text based on a prompt you give it, be it continuing the prompt or responding, and can often give the impression of comprehension of human language and a deeper understanding of the topic at hand. The paper aims to investigate the question "Can LLMs explain the behaviour of other complex predictive models?" by exploiting the in-context learning (ICL) capabilities of **llm!**s. ICL allows **llm!**s to perform well on new tasks by using a few task samples in the prompt. A benefit of using an **llm!** as a post-hoc explainer, is that the output given by the model will already be written in natural language and should be understandable by a layman. The paper presumes that the local behaviour of a model is a linear decision boundary, and by using a sample x and perturbing it to x', and presenting both the samples and the perturbation as natural language we could get an explanation from the **llm!** for the outcome. With a sufficient number of perturbations in a neighbourhood around x the **llm!** is expected to explain the behaviour in this neighbourhood, and rank the features in order of importance.

While the faithfulness of the **llm!** as an explainer is on par with other **xai!** methods used for classification, meaning that the reasons provided are enough to explain the output, I am sceptical of the fidelity of the **llm!** for two reasons. One is the same as for why other **xai!** models often struggle with fidelity, the temporal aspect. If applied in the same way as in the paper it would not consider the intent or the past and only what part of the current observation made it make a certain decision. The other is that I am sceptical of claims presented by an **llm!** in general as these are all just guesses. Good guesses a lot of the time, but still just guesses.

We could however potentially change the implementation so it considers the temporal aspect, and this might be a viable post hoc explainer after some more research into prompt engineering.

2.4.5 ACTER: Diverse and Actionable Counterfactual Sequences for Explaining and Diagnosing Reinforcement Learning Policies

The paper presents ACTER, an algorithm that uses counterfactual sequences with actionable advice on how to avoid failure for an **rl!** policy. It does this by using an ***ea! (ea!)*** known as ***nsga! (nsga!)*** to generate counterfactual sequences that don't lead to failure as close as possible to factual sequences that lead to failure. This paper presents counterfactual sequences and not just actions, which means it also presents how to avoid the state that lead to failure to begin with, which should, if ACTER is implemented correctly, allow us to significantly reduce the amount of times our policy fails. It also offers multiple counterfactuals to allow the end user to decide which counterfactual is preferable to their use case.

There are 4 hypothesises tested by the paper. The last two considers laymen users and are therefore not as interesting to us. The first two however "ACTER can produce counterfactual sequences that prevent failure with lower effort and higher certainty in stochastic environments compared to the baselines." and "ACTER can produce a set of counterfactual sequences that offer more diverse ways of preventing failure compared to the baselines."

are partially and fully confirmed respectively. Which means ACTER will likely be a useful tool to explain and diagnose our **rl!** policy.

2.5 Related Works

This section describes recent development in the field of planning and future predictions. All the papers described here are directly relevant to the methods i developed.

2.5.1 Predicting Future Actions Of Reinforcement Learning Agents

This paper describes a method developed by the author that predicts future actions and events [**chung2024predictingfutureactionsreinforcement**]. It does this for non planning policies, like Impala, implicit planning policies like **drc!**, see section **??**, and explicit planning policies like MuZero and Thinker.

Non planning policies are policies designed without planning in mind. Pure **nn!** policies without recurrency are considered to be non planning, as they have not been found to exhibit planning like behaviour. This paper found that when predicting future states and actions both inner states from implicit planners, and explicit planners improved accuracy over non planning agents.

Explicit planners are agents who simulate future states with a world model before making a decision on what action to take. They rely on the world model being accurate, and so does using them to predict future states and actions. In some environments learning an accurate world model is not feasible, an example of this is autonomous driving.

The paper proposes two methods on predicting future states and actions, simulation based and inner state based. The inner state approach is most effective when working with explicit planners, but also shows significant improvement for implicit planner agents. The simulation based approach performs very well on implicit planner agents, but requires the opportunity to train a decently performing world model. The paper also shows with an ablated world model, the inner state approach performs better.

2.5.2 An Investigation of Model Free Planning

This paper investigates whether planning like behaviour can emerge from model free **rl!**, without any explicit planning methods or inductive biases designed to induce planning behaviour [**guez2019investigationmodelfreeplanning**]. The authors introduce and explore if the **drc!** architecture can learn to implicitly plan through training. World models suffer from scalability issues, and inductive biases require a priori knowledge of the environment. The **drc!** architecture is comprised of 3 layers of ConvLSTMs and each layer is iterated through 3 times each at each timestep.

This architecture is tested on domains that require planning such as Sokoban, and they test the agents ability to generalize as well as data efficiency. The results of the paper is that not only does the model exhibit planning like behaviour, but it outperforms state of the art methods that use inductive biases or world models. The paper described in section **??**

finds that using the inner state of implicit planner agents improves predicted future states and actions.

Chapter 3

Methodology

This chapter will elaborate on the methods I will use to perform the experiments. Firstly, in section **??**, I will discuss in detail the state and events we want to predict. Then, in section **??**, I will elucidate how i attribute importance to observation features. After that, in section **??**, I will define what I mean by intent for an agent in a **rl!** setting. Then, in section **??**, I will discuss what temporal information I will use and how to get it. Then, in section **??**, I will discuss what statistical tests I perform to validate my results. Lastly, in section **??**, I will elaborate on what environments I perform the experiments on, and how these environments are structured.

3.1 Event and State prediction

For the state prediction part of the experiments I will attempt to improve predictions on x and y coordinates of an agent a given time into the future.

For the event prediction part of the experiments I will attempt to improve predictions on whether an agent encounters a critical state in a specified sub-trajectory. This section will define what I mean by critical states.

3.1.1 Critical states

In a safety critical real world scenario, like autonomous driving, its important to identify when a vehicle is about to make an important decision, a decision where a mistake could lead to important consequences. If we can identify these states we could make efforts to increase the likelihood of correct action in these states, either by taking over as a human, or improving the policy in these scenarios.

3.1.2 Maximum logit difference

The output of the policy, in the case of discrete action space, is a vector with size n, where n is the amount of possible actions, without an activation function. This is also known as the logit vector. A state in which there is a high difference between the highest logit and the lowest logit means the agent has learned to avoid the action associated with the low logit and prefers the action with the high logit, which would mean the agent associates the high logit action with high future rewards, and the low logit action with low future rewards. If the logits are close to each other it means the agent is either uncertain about which action to choose or indifferent to the different actions in this state. Given that the policy has been trained well enough that it correctly predicts the optimal action in a state, I consider this state to be critical if the difference between the highest logit and the lowest logit is higher than some chosen threshold. To compute whether a trajectory contains a critical state I consider the maximum logit difference in each of the states, and assign the maximum of these differences to the trajectory. After computing a value for every trajectory, we consider the median of the values assigned to the trajectories as the threshold for whether a state contains a critical state or not. This means we consider half the trajectories to contain a critical state. The main reason for this is it makes using a baseline of random guesses have expected 50% accuracy. This means if we train a binary classifier with initial observation as input and whether the trajectory contains a critical state as output any improvement over 50% means a network has learned a meaningful connection between the input and the output, as opposed to simply learning that one class is more common than another.

$$\begin{gathered}
\mathbf{z_i}(x) = [z_{i,1}(x),\, z_{i,2}(x),\, \ldots,\, z_{i,K}(x)] \\
z_{i,\max}(x) = \max_{j \in \{1,\ldots,K\}} z_{i,j}(x) \\
z_{i,\min}(x) = \min_{j \in \{1,\ldots,K\}} z_{i,j}(x) \\
\Delta(x_i) = z_{i,\max}(x) - z_{i,\min}(x) \\
\Delta(x) = \max \Delta(x_0), \Delta(x_1), \ldots, \Delta(x_K)
\end{gathered}$$

$$\Delta(x) > \tau \tag{3.1}$$

where $z_{i,j}$ is output j of policy π_θ in state i, **??** is the criterion for marking a sub-trajectory as containing a critical state, and τ is the threshold for said equation.

3.1.3 Counterfactual sequences

In the case of continuous action spaces we cannot look at maximum logit difference, as a policy network doesn't output logits, but rather the actions to choose. To discover counterfactual sequences we use the **ea!** known as **nsga!**. This algorithm is a significant improvement over earlier multi-objective EAs that use non dominated sorting. It has a complexity of $O(MN^2)$ instead of $O(MN^3)$, elitist approach and a specified sharing parameter, all three of which, many earlier algorithms lacked. [**Deb2001AFA**] Pseudocode for sorting in algorithm **??**. We use multi objective search because we want to minimize action change while maximizing total reward change throughout a simulation. This defines our two objectives. This way we can discover what actions or combinations there of have the highest impact on the environment. When working with a discrete action space, that is when we have a finite, usually relatively small, amount of actions to choose from, we define the action objective that we want to minimize as how many actions in a single timestep are different from a predefined sequence found with a trained model. After the timestep with the found actions we again use the model to roll out the rest of the sequence. We compare the altered sequence to the original sequence and measure the total reward both of these sequences receive. We want to maximize this difference.

Algorithm 1 Fast Non-Dominated Sort

```
$F_i \leftarrow \emptyset$
**for all** $p \in P$ **do**
    $S_p \leftarrow \emptyset$
    $n_p \leftarrow 0$
    **for all** $q \in P$ **do**
        **if** $p \prec q$ **then**
            $S_p \leftarrow S_p \cup q$
        **else if** $q \prec p$ **then**
            $n_p \leftarrow n_p + 1$
        **end if**
    **end for**
    **if** $n_p == 0$ **then**
        $F_1 \leftarrow F_1 \cup p$
        $p_{rank} \leftarrow 1$
    **end if**
**end for**
$i \leftarrow 1$
**while** $F_i \neq \emptyset$ **do**
    $Q \leftarrow \emptyset$
    **for all** $p \in F_i$ **do**
        **for all** $q \in S_p$ **do**
            $n_q \leftarrow n_q - 1$
            **if** $n_q \leftarrow 0$ **then**
                $q_{rank} \leftarrow i + 1$
                $Q \leftarrow Q \cup q$
            **end if**
        **end for**
    **end for**
    $i \leftarrow i + 1$
    $F_i \leftarrow Q$
**end while**
```

3.2 Feature importance

I decided to experiment on whether including feature importance could improve the event and state predictions. The methods used to attribute importance to features were SHAP described in section **??**, and Integrated Gradients described in section **??**. This section also describes how I decided on choosing baseline values for these methods. I can use these methods either on logit output or on softmax values with different intentions. If I use the methods on the softmax values I get insight into what affects the confidence of the policy, while if I use them on logit output it better reflects the policy networks inner workings, as in my case, the policy network outputs logits and not softmax scores.

3.2.1 Shapley values for observations

The Shapley value is an idea from cooperative game theory where it was used to measure the contribution of each player to the total outcome. It has been adapted to deep reinforcement learning where each feature is considered a player and the policy is considered a game. In our case we initially want to use it to explain actions so we consider that to be the outcome of the game. The Shapley value is calculated by taking each feature and finding the marginal contribution, i.e. seeing how the prediction changes when you include the feature and compare the outcome to when it wasn't included, you do this over all possible coalitions of features. See algorithm **??**. The Shapley value has a computational complexity of $O(2^n)$, and because of this we often use estimates instead of exact Shapley values. There are two main ways of reducing the complexity. The first one is using a surrogate model instead of the policy to reduce the number of features, and the second is reducing the number of coalitions. In our case we reduce the number of coalitions to some number 2^K and use K-means clustering to increase representability and reduce variance. Since we have forward passes of the policy, and each can be considered one example of a game we sample the games and use the average marginal contribution over all the games instead of just one game. The Shapley value can help us understand how the policy acts in general. We can also use it to get explanations for single observations, however these explanations won't always have high fidelity. We will explore more of how the fidelity of Shapley values for single explanations compare to high fidelity explainability methods like gradient methods. See section **??**

3.2.2 Integrated Gradients

By integrating the gradient of the model prediction when going from a specified baseline value as input to our specific observation x we get a high fidelity explanation for what features were important in that specific observation. We integrate from the baseline instead of just getting the gradient in our observation because of the saturation problem. If a feature is important to the action the gradient could be small for the value x, but the integral of the gradient from the baseline to x would still be large. Conversely a gradient for the observation x could be significant in a small area around a feature in x even if that feature isn't as significant as the gradient in x would lead us to believe [**sundararajan2017axiomaticattributiondeepnetworks**]. See algorithm **??**. Comparing figure **??** with figure **??** we can see that the values are similar. This means that in the case of this observation, the fidelity of the Shapley explanation is high.

3.2.3 Choosing a feature attribution baseline

Both the Shapley value attribution and the Integrated Gradients attribution I use require a baseline. A baseline should represents a neutral, or absence of a value, and needs to be from within the domain that a model is trained on, as

Algorithm 2 Shapley Value Calculation with Baseline Replacement

```
Input: Model $f$, input sample $x$, baseline $x'$, set of features $F$
Output: Shapley values $\phi_f$ for each feature $f \in F$
$\phi_f \leftarrow 0 \; \forall f \in F$                                    ▷ Initialize Shapley values
$n \leftarrow |F|$                                                  ▷ Number of features
for all $f \in F$ do
    for all $S \subseteq F \setminus \{f\}$ do
        $x_S \leftarrow \texttt{construct}(x, x', S)$      ▷ Replace features not in $S$ with baseline values
        $x_{S\cup\{f\}} \leftarrow \texttt{construct}(x, x', S \cup \{f\})$
        $\Delta \leftarrow f(x_{S\cup\{f\}}) - f(x_S)$                   ▷ Marginal contribution of feature $f$
        $\phi_f \leftarrow \phi_f + \frac{|S|!(n-|S|-1)!}{n!}\Delta$
    end for
end for
return $\{\phi_f : f \in F\}$
function CONSTRUCT$(x, x', S)$
    Input: Original sample $x$, baseline $x'$, feature subset $S$
    Output: Modified sample $x_S$
    for all $f \in F$ do
        if $f \in S$ then
            $x_S[f] \leftarrow x[f]$
        else
            $x_S[f] \leftarrow x'[f]$                              ▷ Replace with baseline value
        end if
    end for
    return $x_S$
end function
```

Algorithm 3 Integrated Gradients for Feature Importance in Reinforcement Learning

```
Input: rl! policy $\pi_\theta$, input $x$, baseline input $\bar{x}$, number of steps $m$
Output: Integrated gradients $\texttt{IG}(x, \bar{x})$
$\texttt{IG}(x, \bar{x}) \leftarrow 0 \; \forall f \in \texttt{features of } x$          ▷ Initialize integrated gradients to zero
for $i = 1$ to $m$ do
    $\alpha_i \leftarrow \frac{i}{m}$
    $x_i \leftarrow \bar{x} + \alpha_i \times (x - \bar{x})$
    $grad_i \leftarrow \nabla_x \texttt{value}(\pi_\theta, x_i)$            ▷ Compute the gradient of the value function
    $\texttt{IG}(x, \bar{x}) \leftarrow \texttt{IG}(x, \bar{x}) + grad_i \times \frac{(x - \bar{x})}{m}$  ▷ Accumulate gradients scaled by the step
size
end for
function VALUE$(\pi_\theta, x)$
    Input: rl! policy $\pi_\theta$, input $x$
    Output: Model prediction for input $x$
    Feed input $x$ to policy $\pi_\theta$
    return model output
end function
```

a models could behave unpredictably on data outside of their domain, In my case, the environments we used has observation spaces with values in $[0, 1]$, meaning that for our baseline we need to choose values in this range. There are several ways to choose a baseline, and each baseline will yield different results. If the domain is a set of images, a common baseline to use is black, or $(0, 0, 0)$ if the images are represented with RGB values[**baseline_bird**]. Intuitively, this can be understood as **nn!**s looking for patterns in an image, which disappear if the image is monochrome. In my case, I do not have images, but tabular data. Using 0 as baseline values is likely not a good choice, as each value when using tabular data represents an interpretable value and 0 represents a value the model would process as extreme, instead of neutral. Another possibility is choosing the dataset mean as a baseline, which has been used successfully in other **rl!** applications, although not **rl!** applications with tabular data that I am aware of [**baseline_rl**].

3.3 Extracting intent

In an environment where the agents successfully reaches a desirable state we can assume that at some point in the trajectory an agents intent is to reach this state. We can train a prediction model on such an environment that uses a certain state or observation as input and outputs features of interest some time in the future. In the example of autonomous vehicles the output could be predicted position of the vehicle at a future point in time. If our prediction model is successful, then it could be considered part of an explanation for the intent of an agent. Additionally, we train different models with different sets of inputs to identify what is important to identify intent. We train one model that takes just the observation as input, one that takes observation and action, and one that takes observation and action one hot encoded. While action as an integer and action one hot encoded contains the same amount of information it could be easier for a network to learn correlations with the additional inputs. Looking at the loss curve during training, as well as final loss, this suspicion is confirmed to be true when dealing with a network architecture and environment like we are currently using, 2 fully connected hidden layers with the hyperbolic tangent as the activation, in the environment simple_spread_v3. While the input layer for the three prediction models are slightly different the difference in training time is negligible. After 200 epochs the base prediction model reached a evaluation loss of 0.0407, the model with integer action as part of the input reached a evaluation loss of 0.0359 and the model with one-hot encoded action as part of the input reached an evaluation loss of 0.0322. See figure **??**, **??** and **??**. Both the base model and the one-hot action model stagnate long before 200 epochs is reached. If we let the model with integer action as input train for another 50 epochs it does not improve. The test loss for these three models respectively is ... . These graphs and values indicate that while action and an explanation are useful for extracting intent, in the context of our environment and model one-hot values and integrated gradient explanation are the most useful. The difference between using Shapley values and

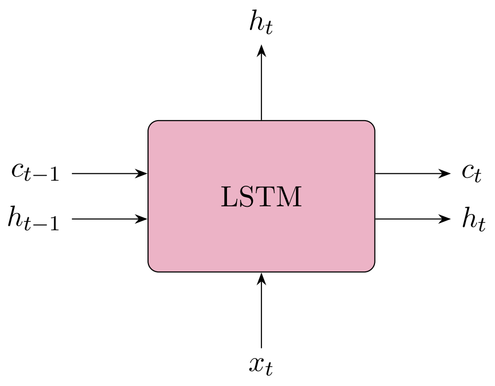


Figure 3.1: Simplified illustration of an **lstm!** layer

3.4 Temporal information

In this section I define what I refer to as temporal information and define the methods I use to extract this information, as well as why I do it.

3.4.1 LSTM values

An **lstm!** cell takes as input a cell state c_{t-1}, a hidden state h_{t-1}, and an input X_t for each timestep. It outputs a cell state c_t, a hidden state h_t and an output y_t. h_{t-1} is concatenated with X_t and then with trainable weight matrices $\mathbf{W}$ and biases b, c_t and h_t are computed according to the following equations:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot \tanh(c_t)$$

Intuitively, f_t can be viewed as how much of the previous cell state c_{t-1} to forget, i_t can be viewed as how much of the candidate cell state $\tilde{c}_t$ to remember, and o_t can be viewed as how much of the c_t to output. Capital letters represent matrices and lowercase letters represent vectors.

As c_t and h_t are helpful when predicting future actions and states [**chung2024predictingfutureactionsreinforcement**] when considering the **drc!** architecture, I hypothesize that the same holds when using a regular **lstm!** layer. See figure **??**.

3.4.2 Frame Stacking

Frame stacking in **rl!** is a technique used to provide an agent with temporal context when tasked with choosing an action by combining the previous n observations into a single input to the policy. In some environments an observation at a single timestep isn't enough to capture the full state. In the case of autonomous vehicles a single observation might not have information on whether a car is standing still or in motion, and the velocity of a car is important to take into account when making an action. If we have an observation space of 18, and we want to stack the previous 4 frames to feed into a fully connected input layer to a **nn!** we would insted of having the input o_t, where o_t is a vector of size 18, we would instead get

$$s_t = [o_t, o_{t-1}, o_{t-2}, o_{t-3}]$$

which is a 18×4 matrix, which we flatten into a single vector of size 72. If we instead of having a fully connected input layer, have a **cnn!** or another architecture that takes spatial context into consideration when processing an input, we would not flatten it. Frame stacking does not give the **nn!** architecture memory, however it does provide the network with short term temporal context. I hypothesize that for event and state prediction for an agent, having information about previous observations and their transitions is helpful for predicting how the state will change in the future. In the case of autonomous vehicles its natural to assume that knowing the speed of which a vehicle has been moving will be relevant when trying to predict where it will be n steps into the future.

3.5 Statistical tests

The null hypothesis chosen, H_0, is that the method provides no change in expected performance. To confirm that the difference in performance between baseline and our method is statistically significant I performed an independent one-tailed t-test, i.e. I calculated the likelihood that the increase in observed performance, or a higher increase in performance, happened by chance, and not because the true mean of the performance of the developed method is increased from baseline. We bootstrapped the full dataset several times, i.e. created new datasets by sampling with replacement from the original datasets, and trained my prediction **nn!**s on every dataset. From this we get a distribution of performances that we can use to perform a t-test and get a p-value. If this p-value is lower than a specified threshold value α I consider the result to be statistically significant. I set $\alpha = 0.05$.

Formally:

$$H_0 : \mu = \mu_0$$

where μ is the mean of the method i developed, and μ_0 is the baseline mean.

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

where $\bar{x}$ is the mean of the samples for the method i developed, s is the standard deviation for the samples in the population and n is the size of the population.

For a right tailed t-test, which i use to measure whether the accuracy for event prediction has improved:

$$p = P(T \geq t) = \int_{t}^{\infty} f_T(u)\, du$$

and for a left tailed t-test, which i use to measure whether the error for state prediction has decreased:

$$p = P(T \leq t) = \int_{-\infty}^{t} f_T(u)\, du$$

$$\text{Reject } H\_0 \quad \text{if} \quad p < \alpha$$

3.6 Environments

For a better understanding of what state and event predictions refer to in my experiments, it is important to understand what type of environments the methods to extract states events are used in. In section **??** we describe the Simple Spread environment and in section **??** we describe the Knights Archers Zombies environment.

3.6.1 Simple Spread

In the Simple Spread environment there are n agents, and n landmarks. Simple spread is a cooperative environment where the goal of an agent is for all the agents to reach a landmark, without colliding with the other agents. The agents are modeled as circles and the landmarks as dots. This environment was chosen partially for its simplicity, and the intuitive understanding of what features humans would consider important in such a setting. Developing our methods for this environment should also be relatively efficient due to the observation space of size 18, which is smaller than most other PettingZoo environments, and as such training the prediction **nn!**s and calculation of feature importance is less computationally expensive. The global reward, R_g, is the negative total distance from each landmark to the closest agent measured with euclidean distance and the local reward, R_l, is -1 for every agent a given agent is colliding with.

Both continuous and discrete actions are available, I went with discrete. In the discrete action space each agent has the ability to idle, move up, move down, move left, or move right. I used $n = 3$ and local ratio 0.5, i.e. the reward, R, for an agent is $R = 0.5 \times R_g + 0.5 \times R_l$. See figure **??**

A critical state are states where the agents actions strongly impact returns. As returns are correlated with rewards a critical state is likely a state where agents might collide or make significant progress on getting closer to the landmark.

3.6.2 Knights Archers Zombies

The KAZ environment has a maximum of n archers, a maximum of m knights, and a maximum number of zombies k. The environment initializes with n archers, m knights and 0 zombies. The goal is for the archers and knights

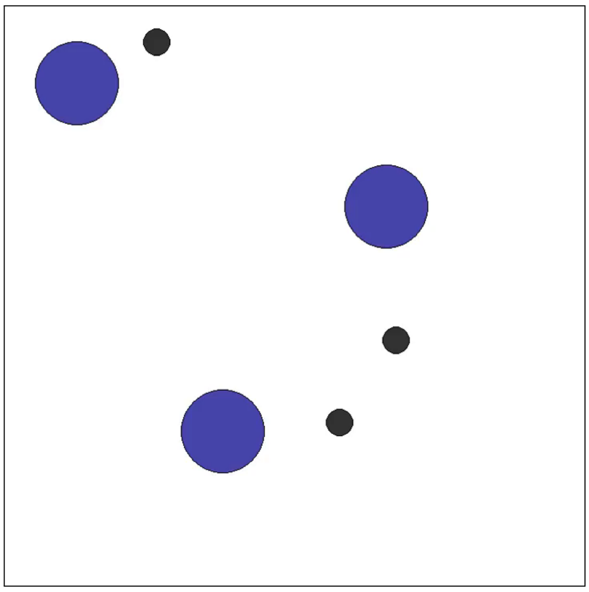


Figure 3.2: Rendered frame from the Simple Spread environment. Agents are rendered as big blue circles, landmarks are rendered as small gray circles

to kill zombies. Zombies spawn at the top border, at a set spawn rate, and travel down the screen moving randomly to the left or right. If an archer hits a zombie with an arrow it gets a reward of 1, and if a knight hits a zombie with a mace it gets a reward of 1. Once a zombie gets hit it dies. If a zombie collides with an agent the agent dies. The game ends once all agents are dead, a zombie reaches the bottom of the screen or some max length is reached.

For my experiments I used 2 archers, 2 knights, and a maximum of 10 zombies. The action space is discrete and allows the agents to idle, move forward, rotate clockwise, rotate counter clockwise, and attack. The observation space is either image based or vector based. I will use vector based for my experiments, which makes it a size of 135. See figure **??**

In the Knights Archers Zombies environment a critical state might be a state where an agent could die or is about to shoot an arrow that might kill a zombie, as these states are associated with changes in returns, immediate in the case of killing a zombie and delayed lower returns in the case of dying as an agent, as this does not incur a penalty, but it does make the agent unable to gain future rewards.

3.7 Policy creation

This section will describe the details of how we create the policies for the environments.

3.7.1 Reinforcement learning library

There are a lot of options when deciding what library to use when designing **rl!** policies. I decided to use the library RLlib by Ray. The main benefits of this library is ease of implementation when working on heterogeneous agents,

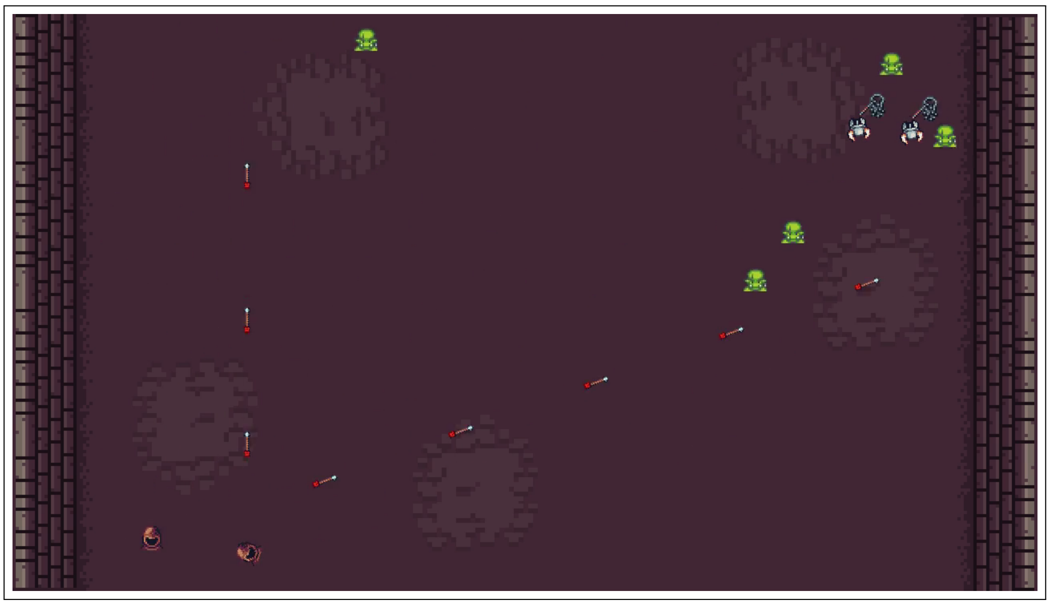


Figure 3.3: Rendered frame from the Knights Archers Zombies environment.

as each type of agent requires a specifically trained network to perform well. It also has scalability [**rayrllib**]. It is required to train several policies for the creation of this thesis, therefore scalability when optimizing a policy is helpful.

3.7.2 Policy construction

I used a **ppo!** algorithm when optimizing my policy, because this method outperforms other algorithms in several environments in the continuous observation domain [**schulman2017proximalpolicyoptimizationalgorithms**], as well as the fact that is it applicable to **marl!** systems. The policy architecture consisted of an input layer with the observation of the environment as the size, two hidden layers of size 64, the hyperbolic tangent function, see **??**, colloquially known as tanh as the hidden layer activation function. This function is used for its "s-shape", bounding values from $y = -1$ to $y = 1$, see figure **??**. The output layer has the size of the action space of the agent.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.2}$$

3.7.3 Training

The training was done with a batch size of 1024, for stability, learning rate and gamma was automatically tuned for the policies in the Knights Archers Zombies environments, but set to $1e - 4$ and 0.8 in the Simple Spread environment as i found these to reach close to a global optimum for every variation of the environment I trained on, e.g. with or without frame stacking.

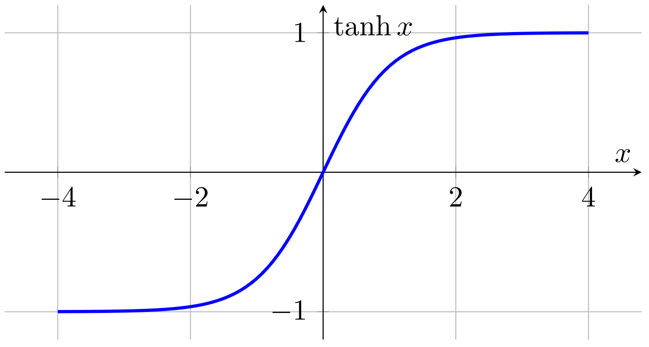


Figure 3.4: Graph showing the hyperbolic tangent function

Chapter 4

Experiments

This chapter will discuss the details on how i perform the experiments, as well as discussing and analyzing the results of these experiments. The experiment described in **??** will be about detecting the effects of actions and feature importance and their ability to elucidate the intent of a policy as aforementioned, the experiment described in **??** explores if and how temporal information can be used to explain the intent of a policy, and the experiments described in **??** explores how to alter the event and state prediction networks to improve predictions.

4.1 General experimental setup

I have several **nn!**s for each experiment. The **nn!**s have two hidden layers of size 64 each, the activation function $tanh$, and outputs either part of the state, (x, y) coordinates for an agent for state prediction, or a single value, whether the trajectory contains a critical state or not, for event prediction. The experiments attempt to test whether my developed methods have an effect on event and state predictions, how impactful these effects are and what contributes to whether the effect is significant or not.I trained them with a scheduler which would divide learning rate by 3 from an original of $1e-4$, down to a minimum of $1e-6$ whenever the calculated loss on the validation set stopped decreasing. All training was stopped after 200 epochs.

4.1.1 State prediction neural nets and datasets

For the state prediction I initially trained a **nn!**, $state_{base}$ to take the observations made by a type of agent from step n, Obs_n, and output the x and y coordinates of the same agent at step $n + m$ in a trajectory. The dataset D_{pos} was constructed such that each data point was from a different trajectory τ, to make sure all the data points were independent of each other, in order to make sure the **nn!** did not learn any such connection. Each data point is a pairing of values $(Obs_n, (x_{n+m}, y_{n+m}))$. The variable n was chosen randomly for each trajectory while m stayed as a constant, in my case $m = 10$. I would for example have 50000 trajectories, each trajectory contains 2 archers, so we would have a dataset of size 100000. 100000 observations and their corresponding positions 10 steps into the future. We assume that both archers use the same policy as they are homogeneous.

4.1.2 Event prediction neural nets and datasets

I also trained a **nn!** for event prediction, $event_{base}$. The event we trained a network to predict was whether an agent would encounter a critical state or not. See section placeholder. The dataset D_{crit} was similar to D_{pos}, however instead of positions, it would contain whether the agent would encounter a critical state within the next 4 steps, $(Obs_n, crit)$, where $crit$ is a binary value, 0, for non critical states, or 1, for critical states. I consider the accuracy of $event_{base}$ and the average error measured in euclidean distance of $state_{base}$ to be the baselines i compare results against in the first experiment. I will train other neural networks.

4.1.3 Baseline performance

$event_{base}$ got an accuracy of 0.8244 on the test set and for $state_{base}$ the average distance from target was 0.0409 on the test set. These **nn!**s are purely post hoc explainers and used no information about the policy to explain the agents intention. As aforementioned we want to include information about the policy to discover how we can improve this prediction, and better discover the intent of the agent.

Only The dataset, and therefore also the input layer, was different between the runs. I trained the networks for a total of 200 epochs, after which improvement was negligible, if at all existent, with a learning rate scheduler that decreased learning rate when loss reached a plateau, which was measured on a validation set.

4.2 Action and explanation inclusion

As aforementioned this experiment explores how including action and explanations improves event and state predictions on the Simple Spread and Knights Archers Zombies environment.

4.2.1 setup

I enhanced both the dataset D_{pos} and D_{crit} with a series of other data. For both datasets I included the action, encoded by an integer from 0 to n_{acts}, i.e. the amount of actions an agent has available, taken at Obs_n for each data point, to use as input for an **nn!**. I then trained two **nn!**s with the same architecture as $state_{base}$ and $event_{base}$ respectively, and once again measured accuracy for the event predictor net with included action, and average error for the state predictor net with included action. If the predictions improved I considered the action to be useful on its own for the predictions in the given environment.

This section will go into detail on how i included actions and explanations.

One-hot and integer inclusion

I then one-hot encoded the action, i.e. encoded the action as a n_{acts} dimensional unit vector with a 1 in the chosen action dimension. This includes the same amount of information as represented as an integer, however I hypothesized that the resulting network would achieve higher performance with one-hot encoded actions as the distance between independent actions when one-hot encoding the actions are all the same, as opposed to when encoding the actions as an integer when the distance between two actions is not constant. For example, between the action encoded as the integer 2 and the action encoded as the integer 3 the Manhattan distance is 1, but between the action encoded as the integer 2 and the action encoded as the integer 4 the Manhattan distance is 2. When one-hot encoding these actions the Manhattan distance between two distinct actions will in all cases be 2. As in the previous cases I also trained a **nn!** on the datasets enhanced with one hot encoded actions.

Integrated Gradients and Shapley Value inclusion

After creating datasets with actions encoded different ways I also created two feature importance explanations for the action chosen in each of the initial observations. Shapley values and Integrated gradients were the two methods chosen. I decided on 2 feature importance measures that internally uses different methods of calculation, gradients and SHAP, to explore if they would yield different results. The motivation behind including feature importance explanations is that it describes some if the inner processing of the policy, and might therefore elucidate what kind of actions it might take in future states, and therefore include some of the intent. On each of the previous datasets I created new datasets with these two feature importance methods included.

Result presentation

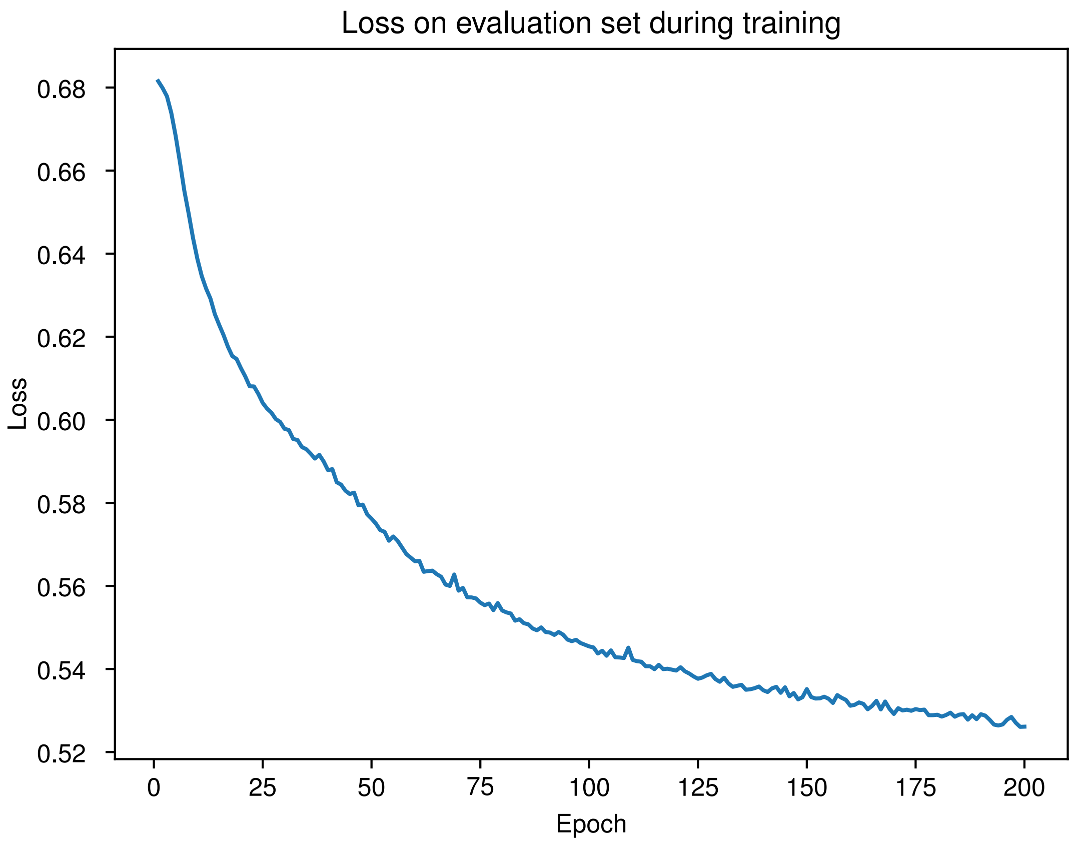
In total I trained 18 neural networks per environment, one for each dataset constructed, and compared the results for accuracy on the event prediction networks and the results for average error on the state prediction networks. I repeated this experiment for a few different environments to explore if the results vary depending on environment.

All results are presented in tables, with asterisks marking significance, and italics used for baseline value.

4.2.2 Results

On state predictions for the Simple Spread environment, the methods I developed are not shown to be an improvement over baseline. It is likely that in the case of predicting positions the observation alone is enough to predict future position, if the policy is close to optimal, or that correlations between explanations and actions, and future position is harder to learn, and would therefore require longer training time, or a larger network. I will investigate this further in experiment.

Table 4.1: Average error for state prediction on the Simple Spread environment without frame stacking, mean over 10 computations



	No explanation	Integrated Gradients	Shapley Values
No Action	*0.145826*	0.149009	0.148133
Integer	0.144696	0.148932	0.148740
One-hot	0.145712	0.148701	0.149457

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

When considering the event predictions for the Simple Spread environment the inclusion of explanations make a noticeably increased difference. With misclassifications being reduced from $1 - 0.700933 = 0.299067$ to $1 - 0.742933 = 0.257067$ when including actions encoded as one hot vectors. A decrease of $1 - \frac{0.257067}{0.299067} = 1 - 0.859563 \approx 14.04\%$. Which is the smallest improvement of the 7 methods that improved the predictions by more than 2%. See table **??**.

Table 4.2: Accuracy for event prediction on the Simple Spread environment without frame stacking, mean over 10 computations

	No explanation	Integrated Gradients	Shapley Values
No Action	*0.711730*	0.806637***	0.793757***
Integer	0.709863	0.810210***	0.800030***
One-hot	0.745763***	0.811403***	0.792647***

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

In the case of state predictions on the Knights Archers Zombies environment the additions of feature importance does not seem to have any important impact as none of the improvements. The inclusion of actions seem to have made an impact on the accuracy of the state prediction, though this decrease in error might just stem from learned knowledge of how an action changes the state, rather than knowledge of how to extract intent from an action. See table **??**.

Table 4.3: Average error, euclidean distance, for state prediction on the Knights Archers Zombies environment

	No explanation	Integrated Gradients	Shapley Values
No Action	0.040901	0.040928	0.040728
Integer	0.039768	0.040216	0.039505
One-hot	0.038924	0.039241	0.038873

In the case of event predictions on the Knights Archers Zombies environment none of the additions seem to have any important impact as none of the improvements. As none of the changes are higher than 2% we do not consider these results relevant. See table **??**.

Table 4.4: Accuracy for event prediction on the Knights Archers Zombies environment

	No explanation	Integrated Gradients	Shapley Values
No action	0.824400	0.829300	0.820650
Integer	0.822700	0.828850	0.819250
One-hot	0.824800	0.826700	0.822950

4.2.3 Analysis

About the figures

In all figures for feature value attribution on the event prediction and state prediction networks, only the 15 most influential features are included, to limit the size and density of the plots. Its natural to consider the features not included as not very influential for the predictions made.

Event prediction baseline

For event prediction in the simple spread environment, we can see that the placement of the landmarks is gets the highest mean of feature attribution values for predicting whether an agent will encounter a critical state or not. Interestingly, with the exception of comms, which were turned off so they are expected to be irrelevant for predicting events, the features with the lowest attributed values were all current agent positions. Since the rewards are structured so that collisions between agents are penalized, it would be natural to expect that positions would be important when considering large differences in logits, that does however not seem to be the case, especially interesting is that the agents own position is the least influential factor when predicting whether the agent is about to encounter a critical state. Intuitively you could consider the position of the landmarks to be important as two landmarks in close proximity to each other might be challenging to reach by two separate agents without those agents colliding with each other. Figure **??** Shapley value attribution on one baseline event prediction **nn!**.

Event prediction with Integrated Gradients

When including integrated gradients and action encoded as an integer for input to the event prediction networks, the Shapley values paint a picture of how influential the explanation especially is for the predictions. In figure **??** the Shapley values for the integrated gradients have been aggregated together to capture the importance of the entire explanation instead the influence of the explanation of each feature, which would be overshadowed by the observation of the agents importance. From the figure we can see that of the 50 instances plotted the explanation is, even in the case of the lowest Shapley value, more influential in predicting that the agent is about to encounter a critical state that any of the regular observations, even the instances with the highest attribution for regular observation values. This is not the case for predicting when the agent is not about to encounter a critical state, though the mean is still significantly more influential than the most influential instances of regular observations. Compared to the case in figure **??** the variance in importance for observation features is reduced, and the order of the magnitudes seems arbitrary.

4.3 Memory inclusion

4.3.1 Setup

There are two main ways of including temporal information in reinforcement learning policies. The first is to include more than just the last observation as input, known as frame stacking. Another design I implemented, that also includes temporal information for the policy is a design that includes an **lstm!** layer in that takes input from the second hidden layer and outputs to the last fully connected layer.

We once again construct datasets which contain pairs on the form (obs, pos) and $(obs, crit)$ respectively. For the observations for the policy using an **lstm!** we also include the hidden values and cell states, (h, c), as part of the input for the event prediction and state prediction **nn!**s, to include the temporal information used by the policy for the prediction **nn!**s, similarly to in the case with frame stacking.

I will once again train 18 neural networks for each environment and see how including the hidden values and cell states of the **lstm!** affect the predictions.

4.3.2 Results

When we include frame stacking in the Simple Spread environment in our case we include the previous 3 observations as well so the total observation space has size $18 \times 4 = 72$. This auxiliary information requires a new policy to be trained as the representation of the environment is different from the case without frame stacking. The baseline results are unchanged from the case without frame stacking, and all other improvements are less pronounced. See table **??**.

Table 4.5: Average error for state prediction on the Simple Spread environment with frame stacking

	No explanation	Integrated Gradients	Shapley Values
No action	0.303008	0.304275	0.298301
Integer	0.289100	0.305222	0.291256
One-hot	0.305099	0.305001	0.298962

For event prediction on the Simple Spread environment with frame stacking the results are improved for no explanation included, and improved in some cases when considering explanations as well, from the results in the case without frame stacking. See table **??**.

Table 4.6: Average accuracy for event prediction on the Simple Spread environment with frame stacking

	No explanation	Integrated Gradients	Shapley Values
No action	0.737900	0.80300	0.756633
Integer	0.724767	0.804433	0.763100
One-hot	0.766567	0.812067	0.780700

Table 4.7: Average error for state prediction on the Simple Spread environment with **lstm!** output and cell states

	No explanation	Integrated Gradients	Shapley Value
No action	0.306358	0.304827	0.305336
Integer	0.306486	0.304529	0.306255
One-hot	0.307398	0.305968	0.305619

Table 4.8: Average accuracy for event prediction on the Simple Spread environment with **lstm!** output and cell states

	No explanation	Integrated Gradients	Shapley Values
No action	0.730967	0.775767	0.739200
Integer	0.715100	0.778400	0.741933
One-hot	0.728867	0.779600	0.742567

4.3.3 Analysis

4.4 Hyperparameter Exploration

4.4.1 Setup

4.4.2 Results

4.4.3 Analysis

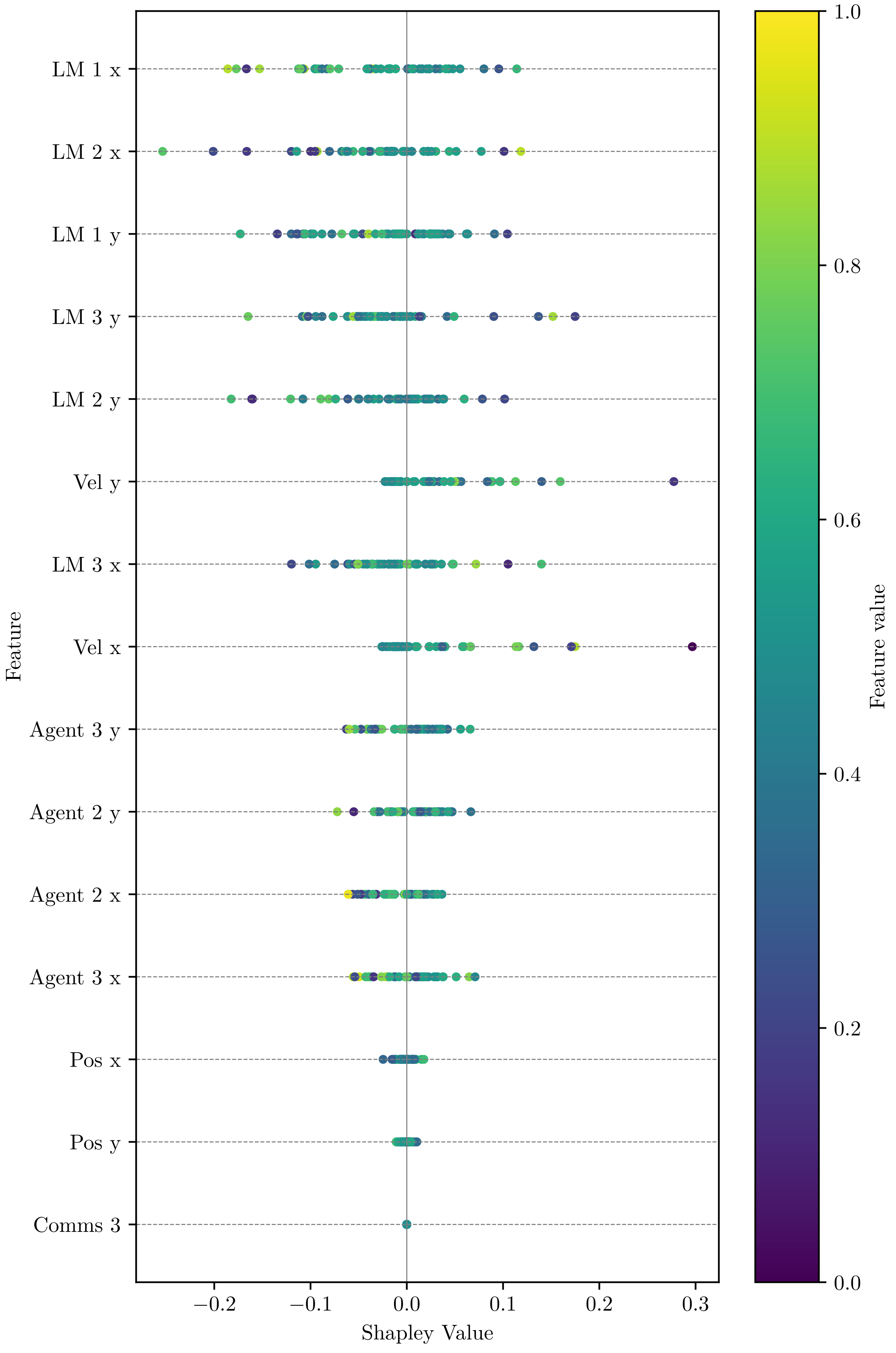


Figure 4.1: Shapley Values on event prediction in the Simple Spread environment without included action or explanation. LM = Landmark, Vel = Velocity, Pos = Position.

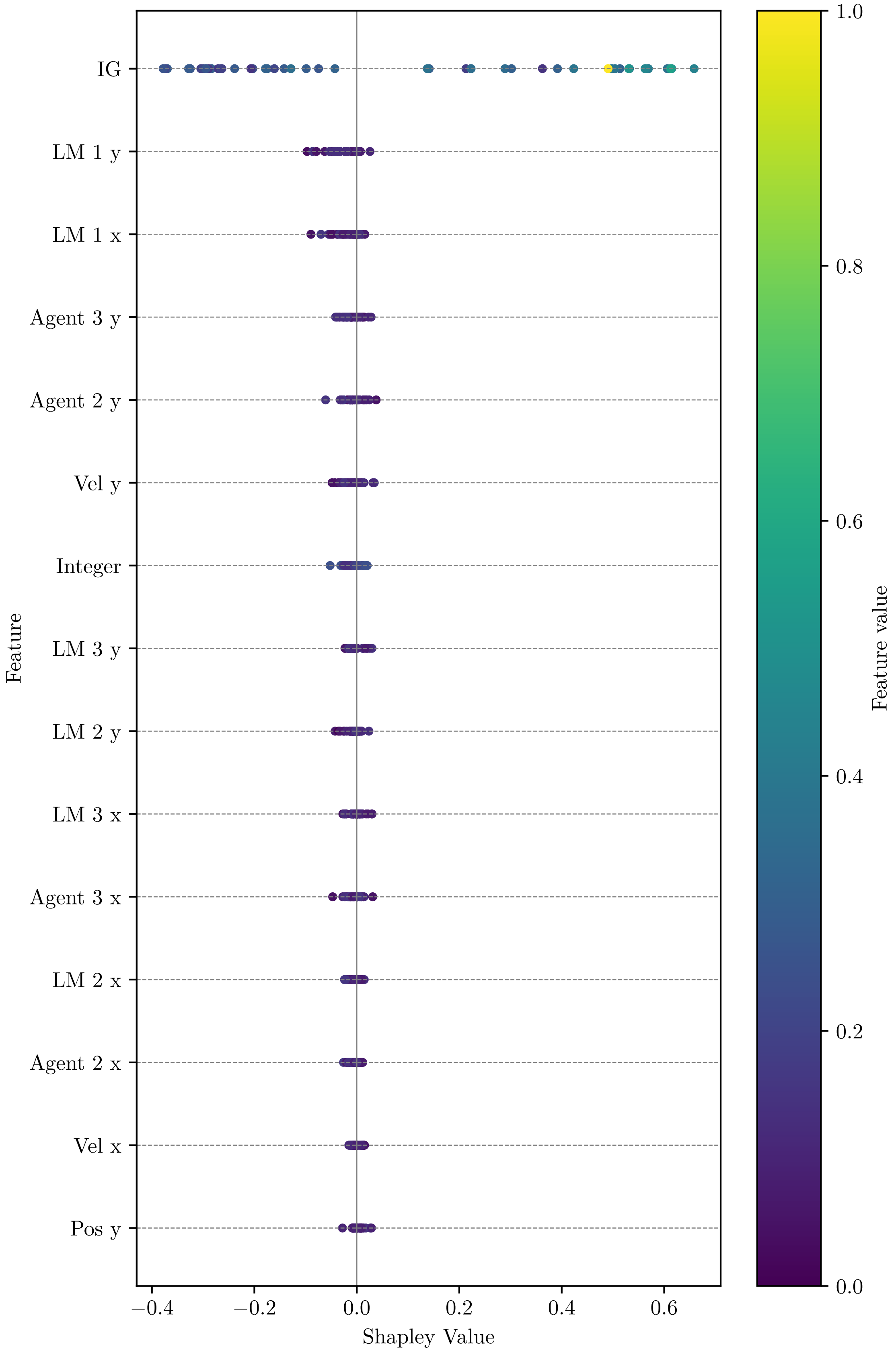


Figure 4.2: Shapley Values on event prediction in the Simple Spread environment with included action represented as an integer, and Aggregated shapley values for Integrated Gradients explanations. IG = Integrated Gradients, LM = Landmark, Vel = Velocity, Pos = Position.

Chapter 5

Discussion

5.1 Conclusion

5.2 Limitations

5.3 Future work